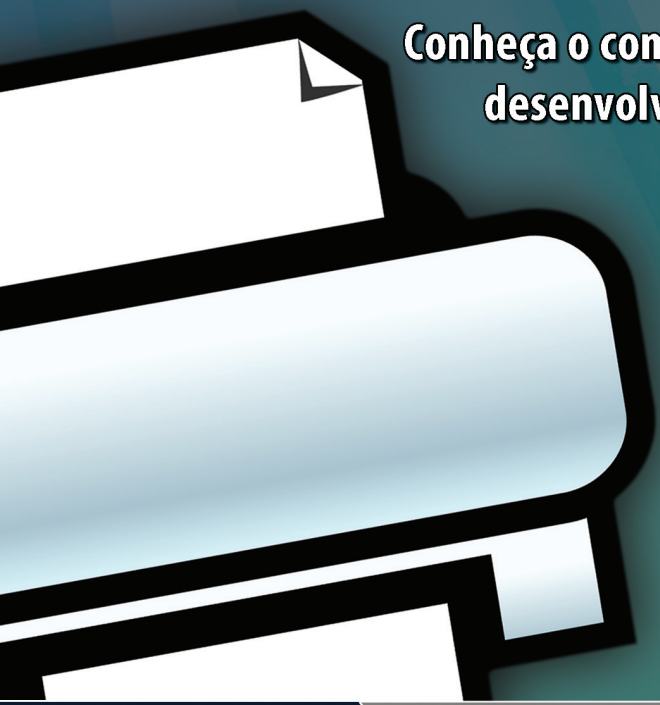
Clube Delphi + PHP

DevMedia group

Ano 9 - Edição 108 - R$ 11,90

Nota Fiscal Eletrônica

Conheça o componente ACBrNFe e desenvolva facilmente a NF-e



Ask the Expert
Alinhar células em um TStringGrid

Quick Update
Ferramentas Case:
Maker, você conhece?

Easy Delphi
Introdução a POO com
Delphi desde o mais básico
até o desenvolvimento avançado

Easy Delphi
Conheça o ReportBuilder e domine
o desenvolvimento de relatórios

Expert
Saiba tudo sobre as classes
Tcollection e TCollectionItem

Web/ Novidades
Domine o Delphi Prism
e crie aplicações robustas com ele

PHP
POO, DAO, PHP e MySQLI, mais
força para suas aplicações PHP

ISSN 1517990-7

Na Web :-

Confira no portal ClubeDelphi PLUS 3 vídeo-aulas:

- Criação de classes em POO no Delphi
- Herança Visual
- Usando propriedades na Orientação a Objetos

SISFÁCIL 2.0

FICOU AINDA, MAIS FÁCIL ADMINISTRAR SEU SITE

Enquete, Fotos, Notícias, Blogs e muito mais.

DISPONÍVEL PARA CLIENTES RJHOST

Seu site pronto e completo

www.sisfacil.com.br

RJHOST

Sumário

Olá, eu sou o **DevMan**! Desta página em diante, eu estarei lhe ajudando a compreender com ainda mais facilidade o conteúdo desta edição. Será um prazer contar com sua companhia! Confira abaixo o que teremos nesta revista:

Boas Práticas — Boas Práticas

12 – ACBrNFe – Componente para Nota Fiscal Eletrônica
Fácil, Rápido e Open-Source
[*André Ferreira de Moraes*]

Delphi — Expert

20 – Padrão TCollection no Delphi
Implementando Coleções de Objetos com TCollection
[*Giuliano Prado*]

Delphi — Novidades/ Tutorial

26 – Curso de Delphi Prism – Parte 1
Novidades da Linguagem – Variáveis Inline e Type Inference
[*Guinther Pauli*]

Easy Delphi — Easy Delphi

32 – ReportBuilder
Montando relatórios com o ReportBuilder
[*Thales Eduardo*]

Easy Delphi — Easy Delphi

38 – Introdução à POO
Aprenda, de uma forma simples, a programar da forma correta
[*Guinther Pauli*]

PHP — PHP

43 – POO, PHP, DAO e MySQLI
Conheça os princípios básicos da POO e veja um exemplo real de aplicação
[*Rafael Silva*]

Você percebeu os ícones ao lado de cada matéria? Eles indicam o que você vai encontrar no artigo – dessa forma, você também pode ter uma idéia geral do que vai encontrar nesta edição como um todo! Os editores trabalham sempre no sentido de fechar a revista seguindo esta definição, para oferecer a você o melhor conteúdo didático!

Confira abaixo a lista com a definição dos tipos de artigo encontrados nesta edição:

[Boas Práticas] Um dos objetivos da revista é levar para o leitor não somente as melhores técnicas mas também as melhores práticas. Esse tipo de artigo foca em técnicas que poderão aumentar a qualidade do desenvolvimento de software.

[Boa Idéia] Esse tipo de artigo é quase um artigo "Mão na Massa" - a diferença é que o exemplo do artigo é tão criativo que é considerado uma 'boa idéia', dentro do contexto de desenvolvimento de software.

[Expert] Artigo com foco no leitor avançado.

[Novidades] Artigos sobre tecnologias de ponta, que ainda não fazem parte o dia-a-dia do desenvolvedor mas que prometem ser a próxima febre.

[Easy Delphi] Artigo com foco no leitor iniciante.

[PHP] Artigo com foco no PHP puro ou Delphi for PHP.

Clube Delphi +PHP

Ano 9 - 108ª Edição 2009 - ISSN 1517990-7 Impresso no Brasil

Corpo Editorial

Editor Geral
Adriano Santos
adriano.santos@devmedia.com.br

Editor Técnico
Paulo Quicoli
pauloquicoli@devmedia.com.br

Daniel Sobrinho
daniel.sobrinho@devmedia.com.br

Comissão Editorial
Fabrício Desbessel, Paulo Quicoli, Adriano Santos e Guinther Pauli.

Diagramação e Capa
Romulo Araujo
romulo@devmedia.com.br

Coordenação Geral
Daniella Costa - daniella@devmedia.com.br

Revisão e Supervisão
Thiago Vincenzo - thiago.v.ciancio@devmedia.com.br

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Atendimento ao Leitor
A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Deise Aleis – Atendimento ao Leitor
www.devmedia.com.br/central/default.asp
(21) 3382-5025

Kaline Dolabella – Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 3382-5025

Publicidade
Para informações sobre veiculação de anúncio na revista ou no site e para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

Kaline Dolabella
publicidade@devmedia.com.br

Fale com o Editor!
É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Adriano Santos - Editor da Revista
adriano.santos@devmedia.com.br

EDITORIAL

Sejam bem vindos a mais uma edição da revista Clube-Delphi recheada de novidades e grandes artigos. Esse mês preparamos uma edição pra lá de especial e tenho certeza que você vai gostar.

Logo de início temos uma artigo muito importante do André Moraes que é um dos desenvolvedores do componente ACBRNFe, componente responsável pela geração de Nota Fiscal Eletrônica. Testamos na redação o componente e sem dúvida é uma mão na roda para quem está a procura de uma solução simpática e de fácil aprendizado.

Na seção Expert, o novo colunista e professor Giuliano, manda muito bem em seu artigo sobre as classes TCollection e TCollectionItem. Sem pestanejar, posso dizer que são classes bem importantes no desenvolvimento Delphi. Onde encontrá-las?

Fácil, lembra dos Fields Editor dos componentes DataSet? Muito bem, quando adicionamos um campo ao Fields Editor, podemos clicar nele e mudar algumas de suas propriedades. Isso também acontece com as colunas de um DBGrid, enfim, vários componentes da VCL utilizam as classes mencionadas. Vamos aprender com o Giuliano a usá-las em nossas aplicações.

Retornando em grande estilo, nosso guru Guinther Pauli fala de Orientação a Objetos na seção EasyDelphi em um artigo super introdutório, mas com uma imensa quantidade de informações. Vale a pena conferir. Na sequência ele inicia seu curso de Delphi Prism falando de linguagem, vantagens, e da IDE do novo produto da Embarcadero.

Ainda se tratando de POO, o Rafael Silva mostra na prática como criar uma aplicação usando POO, DAO (Data Access Object) e MySQLI no PHP. Não podia ser melhor, o artigo está bem detalhado e é possível ter uma noção bastante real de como utilizar esses conceitos na prática.

E se você está iniciando e tem dúvidas na criação de relatórios, eis a solução. O colunista Thales Eduardo nos ensina a criar relatórios com um dos editores mais populares entre os desenvolvedores Delphi: o ReportBuilder.

Uma boa leitura, muito sucesso a todos e até a próxima.

Adriano Santos

adriano.santos@devmedia.com.br

Editor Chefe da revista Clube Delphi
Conheça nosso BLOG em *clubedelphiphp.blogspot.com*

ClubeDelphi PLUS

DevMedia group

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS.
Para mais informações sobre o pacote PLUS, acesse:
http://www.devmedia.com.br/clubedelphi/portal.asp

Hospede seus sites de forma simples, rápida e ilimitada.

Plano **Básico**

- 1 domínio principal
- 50 domínios hospedados
- Transferência **ilimitada**
- Espaço **ilimitado**
- 10 contas de e-mail
- PHP, ASP e ASP.NET
- Painel de controle Plesk
- Suporte 24hs

Tudo isso por apenas

R$ **15,90** por mês
pagamento trimestral

Assine agora mesmo e descubra por que Hospedix é a hospedagem de sites preferida entre os profissionais.

- Conheça também nossos planos profissionais com registro de domínio grátis e isenção de taxas anuais enquanto for cliente.

HOSPEDIX
HOSPEDAGEM PROFISSIONAL

www.hospedix.com.br

30 DIAS GRÁTIS

Ao assinar, digite o código de desconto exclusivo para leitores desta revista e ganhe 30 dias inteiramente grátis.

RVSTMEDIA

ClubeDelphi PLUS

Informativo ClubeDelphi

Portal ClubeDelphi

www.clubedelphi.net/portal

+ 600 vídeo aulas e 7 cursos online

Brinde na web desta edição - 3 vídeos! Confira:

3 Vídeos

Criação de classes na POO para encapsular regras de negócio e Usando propriedades na Orientação a Objetos - Veja nessas aulas de Guinther Pauli, como criar classes para encapsular regras de negócios em uma aplicação POO no Delphi. Saiba também como usar propriedades em POO.
*http://www.devmedia.com.br/articles/viewcomp.asp?comp=989&hl=*poo**
*http://www.devmedia.com.br/articles/viewcomp.asp?comp=4386&hl=*OO**

Herança Visual - Aprenda a criar herança visual em sua aplicação, um poderoso recurso do Delphi que minimiza a digitação de código.
*http://www.devmedia.com.br/articles/viewcomp.asp?comp=4384&hl=*OO**

Para acessar os vídeos utilize os seguintes dados
Login: ***DVM.PLS*** *Senha:* ***8jk23o***

Gostou das vídeo aulas? O portal *www.devmedia.com.br* possui mais de **2 mil vídeo aulas** e **dezenas de cursos online** sobre desenvolvimento de software! Agora você pode comprar as vídeo aulas que preferir e fazer sua própria combinação de vídeos! Saiba mais em *www.devmedia.com.br/creditos*

Edições Anteriores da ClubeDelphi

Você pode comprar todas as edições anteriores da ClubeDelphi através do site da DevMedia!

Para isso basta acessar *https://seguro.devmedia.com.br/edicoes_anteriores.asp*

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Para votar, você vai precisar do código e senha de banca desta edição, que é:
Login: **DVM.PLS** Senha: **8jk23o**

Perguntas e Respostas

Alinhando células de um TStringGrid

Perguntas respondidas por Paulo Quicoli
pauloquicoli@devmedia.com.br

Olá a todos. Tenho recebidos muitos e-mails com dúvidas complexas, mas de solução até que simples. Uma delas tem a ver com o alinhamento de células em um controle TstringGrid. Aparentemente a dúvida é bem complexa, mas na verdade em uma rápida busca na internet acabei encontrando uma solução interessante e resolvi publicar para que todos tenham acesso.

Na **Listagem 1** coloquei todo o código que encontrei e achei interessante para que não haja dúvidas. Para testar o exemplo, apenas coloque um TStringGrid no formulário, preencha-o com algumas informações e em seu evento OnDrawCell preencha com o código indicado. Teste a aplicação.

Esse tipo de solução é bastante encontrada na internet nos mais variados sites. Boa sorte e até mais. ●

Listagem 1. Alinhando células

```
procedure TForm1.StringGrid1DrawCell(Sender: TObject; ACol,
ARow: Integer;
  Rect: TRect; State: TGridDrawState);

 procedure WriteText(StringGrid: TStringGrid; ACanvas: TCanvas;
const ARect: TRect;
    const Text: string; Format: Word);
  const
    DX = 2;
    DY = 2;
  var
    S: array[0..255] of Char;
    B, R: TRect;
  begin
    with Stringgrid, ACanvas, ARect do
    begin
      case Format of
        DT_LEFT: ExtTextOut(Handle, Left + DX, Top + DY,
            ETO_OPAQUE or ETO_CLIPPED, @ARect, StrPCopy
             (S, Text), Length(Text), nil);

        DT_RIGHT: ExtTextOut(Handle, Right - TextWidth
         (Text) - 3, Top + DY,
            ETO_OPAQUE or ETO_CLIPPED, @ARect, StrPCopy
             (S, Text),
            Length(Text), nil);

        DT_CENTER: ExtTextOut(Handle, Left + (Right - Left -
         TextWidth(Text)) div 2,
            Top + DY, ETO_OPAQUE or ETO_CLIPPED, @ARect,
            StrPCopy(S, Text), Length(Text), nil);
      end;
    end;
  end;

  procedure Display(StringGrid: TStringGrid; const S: string;
   Alignment: TAlignment);
  const
    Formats: array[TAlignment] of Word = (DT_LEFT, DT_RIGHT,
     DT_CENTER);
  begin
    WriteText(StringGrid, StringGrid.Canvas, Rect, S,
     Formats[Alignment]);
  end;
begin
  if ACol in [0..2] then
    Display(StringGrid1, StringGrid1.Cells[ACol, ARow],
      taRightJustify)
if ARow = 0 then
      Display(StringGrid1, StringGrid1.Cells[ACol, ARow],
        taCenter)
  end;
```

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:
www.devmedia.com.br/clubedelphi/feedback

Quick Update:

Ferramentas Case: Maker, você conhece?

Saiba um pouco sobre a ferramenta case Maker da empresa Softwell

Adriano Santos

falecom@adrianosantos.pro.br

É desenvolvedor Delphi há mais de 10 anos, programador PHP, especialista em aplicações Win32 e API, bacharel em Comunicação Social. É Editor Chefe das revistas ClubeDelphi e WebMobile (Seção .net). Gerente de Desenvolvimento na SoftPark, membro fundador do DUG-SP (www.dug-sp.com) e mantenedor do blog Delphi to Delphi (www.delphitodelphi.blogspot.com).

Minha experiência de quase 12 anos como desenvolvedor as vezes me trai e acaba me colocando em situações um pouco constrangedoras e desagradáveis. Confesso que sou bem avesso a ferramentas case. Fico imaginando como que o sistema consegue me dar suporte suficiente para criar as rotinas mais complexas de um aplicativo sem eu digitar ao menos um if...then...else. Pois é, sou praticamete um novato no desenvolvimento e já tenho essa cabeça dura e desconfiada, imagine então aqueles que vieram do Assembler, Pascal, Cobol, DataFlex, Clipper e tantas outras linguagens que exigiam do programador muito mais lógica, raciocío e concentração?

Há cerca de alguns meses tive a oportunidade de trabalhar com nada mais nada menos que Dormervilly Tertius que trabalhou durante anos da Borland fazendo história no mercado. Após sua saída da empresa, ele foi trabalhar na empresa Softwell como Diretor Comercial e responsável pelas vendas no Sudeste. Foi ai que conheceu o Maker, uma fantástica ferramenta case desenvolvida, acreditem, em Delphi. Isso mesmo!

Em algumas reuniões com o Tertius ele me apresentou a ferramenta e todo o seu potencial. Realmente é de dar água na boca. O grande Q dela é possibilitar aos desenvolvedores, que nem precisam ser tão desenvolvedores assim, criar fluxogramas que representam as ações do sistema a ser desenvolvido. Em outras palavras, a idéia principal é que o membro da equipe conheça a fundo as regras de negócio do software e a partir delas crie as ações e tarefas que o programa desempenhará. Na **Figura 1** vemos um fluxograma e podemos notar que ele é feito bem a moda antiga mesmo. Caixas de início e fim, balões, caixas de decisão, etc. Feito isso o Maker gera o código referente ao fluxograma e você poderá associá-lo a um botão, evento, ou o que precisar.

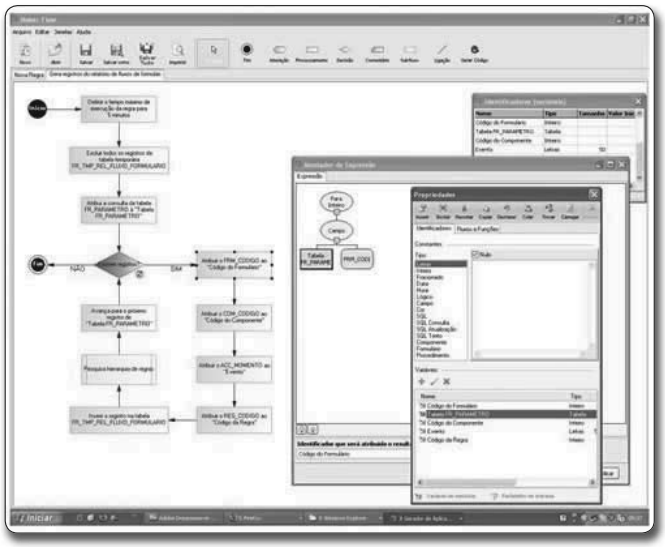


Figura 1. Fluxograma no Maker

Cada interação do fluxograma está ligado a uma espécie de macro. Uma porção de código pré-definida para realizar uma deteminada tarefa, como fazer uma pesquisa no banco, retornar um valor, ou até mesmo checar a impressão digital de uma pessoa. Com isso o desenvolvedor, ou melhor, a equipe ganha em produtividade e melhor ainda, todos entendem facilmente o que o sistema está executando, possibilitando o rápido aprendizado de novos desenvolvedores que venham a integrar a equipe.

Como disse no início, sou avesso a esse tipo de tecnologia, mas confesso que fiquei bastante surpreso com a capacidade do software. Ainda não tive a oportunidade de testar sua performance em produção, porém acredito na tecnologia, pois sei de onde ela vem, Delphi. O Maker é desenvolvido em Delphi e o código-fonte gerado é Java. Exatamente isso que leu companheiro.

O Maker gera pacotes Java compilados que formam todo o sistema. O deploy da aplicação é bastante simples: basta instalar uma máquina virtual do Java e o programa Webrun desenvolvido pela própria Softwell. Copie o pacote .war gerado pelo Maker para o diretório de instalação e pronto, tudo rodando. Para quem deseja trabalhar em Linux, ou mesmo no Windows, mas rodando sob o servidor Apache, é necessário tê-lo instalado e copiar o arquivo .war para o diretório htdocs do Apache. Após isso basta acessar o link: http://localhost:8080/nome-do-seu-sistema que o programa executará normalmente.

Bem, fica a dica aos colegas. Quem desejar testar a solução deverá entrar em contato com a Softwell, pois não há uma versão Trial do sistema. O site é http://www.softwell.com.br.

Um forte abraço a todos e até a próxima. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

A partir de agora você pode contar com a ajuda da

Chegou a Consultoria On-line DevMedia

Consultoria Técnica + Professor Virtual + Certificação

Mais Informações:
www.devmedia.com.br/consultoria_online 21 3382-5025

DevMedia em seus projetos e estudos.

A DevMedia possui um numeroso time de autores, editores e professores que juntos produzem o material que você está acostumado a encontrar em nosso site e revistas. E são exatamente esses mesmos profissionais que estarão a sua disposição para tirar suas dúvidas e ajudá-lo em seus projetos e estudos. Através de uma plataforma 100% web a Consultoria DevMedia garante sigilo absoluto, eficiência e rapidez em todas as respostas. Finalmente você terá ao seu alcance uma consultoria de qualidade por um preço muito acessível. Consulte nossos planos.

Mais um serviço DevMedia group

Nesta seção você encontra artigos sobre técnicas que poderã aumentar a qualidade do desenvolvimento do software

ACBrNFe – Componente para Nota Fiscal Eletrônica

Fácil, Rápido e Open-Source

Nesse artigo veremos
- O que é Nota Fiscal Eletrônica;
- Tipos de Certificados;
- Como instalar o componente ACBrNFe;
- Principais funções do componente ACBrNFe.

Qual a finalidade
Mostrar como o componente ACBrNFe pode facilitar o desenvolvimento de aplicativos para emissão de NF-e.

Quais situações utilizam esse recurso?
Na integração de sistemas para emissão de NF-e.

Resumo do DevMan

Em setembro, com a obrigação de mais de 54 setores a emissão de NF-e, a implementação da NF-e nos sistemas vem atormentando muitos programadores. O componente ACBrNFe pretende facilitar o desenvolvimento, tornado a implementação uma tarefa mais simples e fácil

André Ferreira de Moraes
andre@djsystem.com.br
Trabalha com Delphi e Clipper/xHarbour desde 2001 na DJSystem Informática. Formado em Técnico em Processamento de Dados no Instituto Adventista Paranaense(IAP) e atualmente cursando Sistemas de Informação na Faculdade Santa Bárbara Tatuí – FAESB. Coordena junto com Daniel Simões de Almeida o Projeto ACBr – http://acbr.sf.net.

Nos últimos meses, um assunto muito discutido em todas as empresas é a implantação da Nota Fiscal Eletrônica (NF-e). A partir de abril de 2008, o governo vem obrigando alguns setores a emitirem apenas NF-e. Em abril foram obrigados fabricantes de cigarros, distribuidores de cigarros, produtores, formuladores e importadores de combustíveis líquidos, distribuidores de combustíveis líquidos, transportadores e revendedores retalhistas – TRR; em Setembro de 2008 mais 9 setores passaram a ser obrigados a emitir NF-e; em abril de 2009 mais 25 setores entraram na lista e para setembro de 2009 estão previstos a entrada de mais 54 setores. Por este motivo, se sua empresa vai ser obrigada a emitir NF-e a partir de setembro está na hora de começar a adaptar seu sistema para atender as exigências fiscais, e mesmos os setores que não vão ser obrigados a partir de setembro, o governo espera que se interessem voluntariamente a ser emissor de NF-e.

O que é Nota Fiscal Eletrônica(NF-e)?

O Projeto Nota Fiscal Eletrônica (NF-e), foi criado a partir de reuniões entre os administradores tributários federais, estaduais, do Distrito Federal e dos municípios de capitais. A primeira reunião foi realizada em julho de 2004, onde o objetivo era encontrar soluções conjuntas para uma maior integração administrativa, padronização e melhor qualidade das informações, maior eficácia na fiscalização, entre outros. No final de agosto de 2005, foi assinado o Protocolo ENAT 03/2005 visando à implantação da Nota Fiscal Eletrônica, ficando sobre responsabilidade do ENCAT (Encontro Nacional dos Coordenadores e Administradores Tributários Estaduais) o desenvolvimento do projeto.

A Nota Fiscal Eletrônica é um arquivo digital que contém as informações fiscais sobre a operação comercial, que é assinada digitalmente e transmitida para a Secretaria da Fazenda a qual o contribuinte emitente da NF-e está ligado. Resumindo, a NF-e é um documento digital, emitido e armazenado digitalmente, com o objetivo de documentar uma operação comercial para fins fiscais.

Nota

Apesar da NF-e ser um documento que existe apenas digitalmente, o arquivo deve ser guardado pelo prazo decadencial previsto na legislação, que atualmente é de 5 anos, para apresentação ao fisco quando solicitado.

Certificados Digitais

Um Certificado digital é um arquivo que contém informações da sua empresa como CNPJ, Razão Social, email, informações sobre quem emitiu sua chave pública, número de série do certificado, a validade do certificado e a assinatura da Autoridade Certificadora.

Para NF-e existem dois tipos de certificados disponíveis: e-CNPJ e e-NFe. Com o e-CNPJ além de permitir a assinatura e transmissão dos arquivos da NF-e, também é possível acessar alguns serviços da Secretaria da Receita Federal (SRF), como IRPF - Declarações Entregues, Situação Fiscal do Contribuinte, Cópia de Declaração, Comprovante de Arrecadação, Retificação de Documento de Arrecadação - Redarf Net, Entrega de Declarações com Aposição de Assinatura Digital: Dirf 2005, Simples 2005 e DCTF Mensal, Procuração Eletrônica, Siscomex entre outros. Com o e-NFe apenas é possível utilizá-lo na assinatura e transmissão dos arquivos da NF-e.

Então, qual tipo escolher? O e-CNPJ que geralmente é mais barato e permite acesso a mais serviços ou o e-NFe?

O e-NFe foi desenvolvido para permitir exclusivamente a emissão e gerenciamento da NF-e, enquanto o e-CNPJ permite o acesso a diversos serviços, ou seja, quem tiver acesso ao e-CNPJ terá acesso a serviços que deveriam ser executados apenas pelo representante legal da empresa. Chegamos à conclusão que por questões de segurança deve ser usado o certificado do tipo e-NFe, a não ser que quem emite a NF-e seja também o representante legal da empresa.

Tipo de Certificado: A1 ou A3?

Ao adquirir um certificado digital para a NF-e, a primeira pergunta a ser feita é o tipo de certificado: A1 ou A3?

O certificado do tipo A1 é gerado em um arquivo no micro que foi solicitado e tem validade de um ano. Geralmente, o custo inicial é menor, pois não é necessário nenhum hardware específico para usar o certificado. A principal vantagem do tipo A1 é a possibilidade de criar cópias do certificado, permitindo que seja instalado em mais de um micro. Esta vantagem, pode se tornar uma falha de segurança, caso a cópia seja gerada com uma senha simples, por isso, ao exportar um certificado do tipo A1 escolha uma senha difícil de ser descoberta.

O certificado do tipo A3 é gerado em token ou em um cartão inteligente e tem validade de 3 anos. No caso da versão gerada em cartão inteligente é necessária a compra de uma leitora para utilização do certificado. Por ser gerado em um hardware o A3 é considerado mais seguro, pois não permite a realização de cópias. Por não permitir esta cópia só é possível utilizá-lo quando estiver fisicamente ligado ao computador. Além disso, cada vez que o certificado for conectado ou o micro reiniciado será necessário digitar a senha do certificado. Atenção: caso a senha seja digitada repetidas vezes de forma incorreta o conteúdo do cartão ou token é apagado, inutilizando o certificado.

Recomendo o uso de certificados do tipo A1, ao menos no início do desenvolvimento, pois assim, você poderá ficar com uma cópia do certificado em sua máquina de desenvolvimento para realização de testes, enquanto a outra cópia pode ser usada em produção.

Histórico do componente ACBrNFe

ACBrNFe é um componente pertencente ao Projeto ACBr. O Projeto ACBr é um pacote de componentes open-source, que auxiliam no desenvolvimento de aplicativos comerciais. O objetivo inicial do Projeto ACBr foi criar uma paleta de componentes, com versões VCL e CLX, compatíveis com Delphi, Kylix e Lazarus, que permitem o acesso direto a equipamentos para automação comercial sem a necessidade de DLL's dos fabricantes. O ACBrNFe foi criado a partir da doação de um componente que já era usado em produção, mas que usava soluções proprietárias para assinatura do XML. A partir desta doação, foram desenvolvidas duas versões do componente, uma usando as DLL's capicom, msxml5 e msxml5r, sendo todas da Microsoft. A segunda versão usa o projeto LibXML e OpenSLL, ambos os projetos Open-Source.

Como obter o componente?

Para obter o componente ACBrNFe, primeiramente baixe o TortoiseSVN em tortoisesvn.net/downloads. Após instalar o TortoiseSVN siga os passos abaixo:

- Crie um diretório para baixar o ACBr ;
- Clique com o botão direito sobre o diretório recém-criado e escolha a opção SVN Obter Módulo;
- Na URL do repositório digite https://acbr.svn.sourceforge.net/svnroot/acbr/trunk.

Instalando o componente

Primeiramente, escolha qual versão do componente irá instalar ACBr_NFe_CAPICOM e ACBr_NFe_OpenSSL. As principais diferenças entre as versões são:

ACBr_NFe_CAPICOM - Usa as DLLs da Microsoft CAPICOM (conexão segura) e MSXML (Assinatura e validação do XML) - Apenas funciona no Delphi e Windows. Funciona com certificados do tipo A1 e A3.

ACBr_NFe_OpenSSL - Usa Projetos Open Source: Synapse+OpenSSL (conexão segura) e LibXml2 + XmlSEC (Assinatura e validação do XML) - É compatível com Lazarus, Kylix e com Linux. Funciona apenas com certificados do tipo A1.

Se você pretende desenvolver apenas para Windows e não pretende usar o Lazarus, dê preferência à versão CAPICOM.

Versão OpenSSL:

• Copie a DLL ACBr\ACBrNFe\DLLs\iconv.dll e as DLLs da pasta ACBr\ACBrNFe\DLLs\OpenSSL\ (libeay32.dll, libxml2.dll, libxmlsec.dll, libxmlsec-openssl.dll,libxslt.dll, ssleay32.dll e zlib1.dll) para a pasta Windows\System32;

• Adicione ao Library Path do Delphi as pastas ACBr\ACBrNFe e ACBr\ACBrComum;

• Na IDE do Delphi clique em Tools - Enviroment Options selecione a guia Library, clique no botão elíptico (...) de Library Path, clique no botão elíptico (...) a frente da caixa de texto, localize a pasta ACBr\ACBrNFe no diretório de instalação da Biblioteca, clique em OK, e em Add. Repita o mesmo processo com a pasta ACBr\ACBrComum. Clique em OK para finalizar;

• Na IDE do Delphi, clique em "File - Close All";

• Abra o arquivo ACBr_NFe_OpenSSL.dpk, clique em compile, depois clique em Install.

Versão CAPICOM:

• Copie as DLLs da pasta ACBr\ACBrNFe\DLLs\Capicom (capicom.dll, msxml5.dll e msxml5r.dll) para a pasta Windows\System32;

• Vá ao botão Iniciar do Windows - Executar e execute os seguintes comandos:

```
regsvr32 capicom.dll /s
regsvr32 msxml5.dll /s
```

• Adicione ao Library Path do Delphi as pastas ACBr\ACBrNFe, ACBr\ACBrNFe\win32api e ACBr\ACBrComum

• Na IDE do Delphi clique em Tools, Enviroment Options selecione a guia Library, clique no botão elíptico (...) de Library Path, clique no botão elíptico (...) à frente da caixa de texto, localize a pasta ACBr\ACBrNFe no diretório de instalação da Biblioteca, clique em OK, e em Add. Repita o mesmo processo com a pasta ACBr\ACBrNFe\win32api e ACBr\ACBrComum. Clique em OK para finalizar.

• Na IDE do Delphi, clique em "File - Close All"

• Abra o arquivo ACBr_NFe_CAPICOM.dpk, clique em compile e depois em Install.

Conhecendo as propriedades do componente

Agora que já baixamos e instalamos o componente, vamos conhecer as principais funções do componente. Para testar as funcionalidades do componente é necessário ter um certificado válido disponível.

Primeiro vamos conhecer as propriedades do componente, mostradas na **Figura 1**.

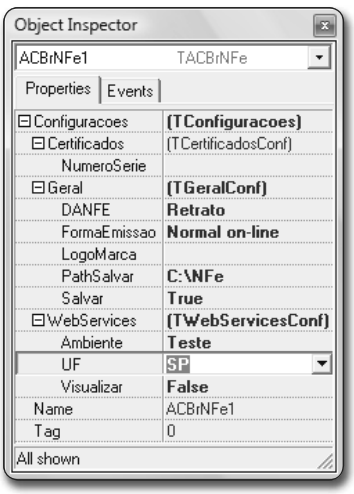


Figura 1. Propriedades do componente

• **NumeroSerie** - Número de Série do Certificado Digital. Para descobrir o número de série do certificado, abra o Internet Explorer, clique no menu Ferramentas - Opções da Internet. Na janela que irá se abrir, clique na aba Conteúdo e no botão Certificados. Dê um duplo clique no certificado, clique na aba Detalhes e selecione o campo Número de Série. Copie o valor deste campo para o bloco de notas, retire os espaços e mude todas as letras para maiúscula. Exemplo: =5a c4 aa 85 39 b3 fc 85 = 5AC4AA8539B3FC85 – este valor deverá ser colocado na propriedade NumeroSerie. Na versão OpenSSL não existe esta propriedade. Para substituí-la, existem duas propriedades com o nome de Certificado, que deverá ser informado o caminho do arquivo .pfx e a propriedade Senha, que é a senha utilizada para proteger o arquivo .pfx;

• **DANFE** - Propriedade para configurar a forma de impressão do DANFE, podendo ser Retrato ou Paisagem;

FormaEmissao - Forma de emissão do XML, podendo ser Normal on-line, Contingência SCAN ou Contingência off-line. Atualmente o componente está apto apenas para a forma Normal on-line, mas em breve serão disponibilizadas as outras formas;

• **LogoMarca** - Caminho para o arquivo com a Logo Marca que será impressa no DANFE. O arquivo deverá ser no formato BMP;

• **PathSalvar** – Caso a propriedade Salvar esteja com o valor True, os arquivos gerados e recebidos pelo componente serão salvos na pasta indicada nesta propriedade;

• **Salvar** – Propriedade booleana que indica ao componente se os arquivos gerados e recebidos deverão ser salvos;

• **Ambiente** – Ambiente que os arquivos serão enviados, podendo ser Teste, que indica o ambiente de homologação, e Producao, que indica o ambiente de produção;

• **UF** – Estado ao que o emissor da NF-e está ligado;
• **Visualizar** – Propriedade booleana que indica se o componente deverá exibir uma mensagem com as respostas recebidas do WebService. Útil no desenvolvimento para ebugar sua aplicação;

Iniciando uma aplicação de teste

Agora que já conhecemos todas as propriedades do componente, vamos iniciar uma aplicação para entender as funcionalidades do componente. Abra a IDE do Delphi e inicie uma nova aplicação, clique no formulário que será criado e modifique as propriedades Name para frmACBrNFe e Caption para Teste das funcionalidades do ACBrNFe, adicione ao form um componente ACBrNFe que está disponível na aba ACBr. Clique no componente ACBrNFe1 e modifique as propriedades NumeroSerie colocando o número de série do certificado e UF para o estado que você deseja testar, verifique se a propriedade Ambiente está com o valor Teste, adicione um componente Memo e altere a propriedade Name para memoResposta, clique no botão elíptico da propriedade Lines e apague todo o conteúdo da janela que será aberta. Por último, adicione 7 componentes do tipo TButton, modifique as propriedades Name e Caption conforme mostra a **Tabela** 1.

Name	Caption
btnStatus	Status Serviço
btnCriarEnviar	Criar e Enviar NF-e
btnConsultar	Consultar NF-e
btnCancelar	Cancelar NF-e
btnInutilizar	Inutilizar Numeração
btnValidar	Validar XML
btnImprimir	Imprimir NF-e

Tabela 1. Propriedades dos botões

Adicione à cláusula uses do seu formulário a unit ACBrNFe-Types para que possamos usar alguns tipos específicos do componente na criação de uma NF-e. Ao terminar as alterações, ajuste seu Form para ficar com a aparência semelhante à **Figura 2**.

Figura 2. Formulário principal

Nota do DevMan

Um arquivo contendo as definições na linguagem XML Schema é chamado de XSD (XML Schema Definition), este descreve a estrutura de um documento.

XML Schema é uma linguagem baseada no formato XML para definição de regras de validação ("esquemas") em documentos no formato XML. Foi a primeira linguagem de esquema para XML a obter o status de recomendação por parte do W3C.

Salve o projeto em uma pasta de sua preferência e copie as pastas Schemas e Report que se encontram dentro da pasta ACBrNFe\ Demo para dentro da pasta do seu projeto. Estas pastas deverão ser distribuídas junto com sua aplicação, pois contém o DANFE e os arquivos XSD fornecidos pelo projeto NF-e usados para validar seu arquivo antes de enviá-lo ao WebService.

Consultar status do Serviço da NF-e

Após os ajustes iniciais vamos começar a testar as funcionalidades do componente. Selecione o botão btnStatus e no evento OnClick adicione o código mostrado na **Listagem 1**.

Listagem 1. Status do Serviço

```
01 ACBrNFe1.WebServices.StatusServico.Executar;
02 memoResposta.Lines.Text := ACBrNFe1.WebServices.
     StatusServico.RetWS;
03 ShowMessage(ACBrNFe1.WebServices.StatusServico.Msg);
```

Como podemos ver na Listagem 1 o componente memoResposta será usado apenas para mostrar o XML de retorno do WebService, a Linha 3 mostrará o resultado da consulta. Ao executar o programa e clicar no botão Status Serviço obteremos o resultado mostrado na **Figura 3**.

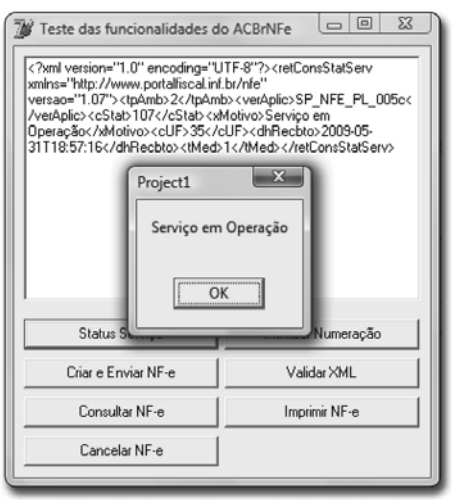


Figura 3. Resultado ao clicar no botão Status Serviço

Criar e Enviar uma NF-e

Para criar uma NF-e e enviá-la, selecione o botão btnCriar Enviar e no evento OnClick adicione o código mostrado na **Listagem 2**.

Listagem 2. Criar e Enviar uma NF-e

```
01  ACBrNFe1.NotasFiscais.Clear;
02
03  with ACBrNFe1.NotasFiscais.Add do
04  begin
05    Identificacao.NaturezaOperacao  := 'VENDA PRODUCAO DO ESTAB.';
06    Identificacao.Codigo            := 1;
07    Identificacao.Numero            := 1;
08    Identificacao.Serie             := 1;
09    Identificacao.DataEmissao       := Date;
10    Identificacao.DataSaida         := Date;
11    Identificacao.Tipo              := tSaida;
12    Identificacao.FormaPagamento    := pgAVista;
13
14    Emitente.CNPJ                       := '00000000000000';
15    Emitente.IE                         := '000000000000';
16    Emitente.Nome.RazaoSocial           := 'RAZAO SOCIAL LTDA EPP';
17    Emitente.Nome.Fantasia              := 'NOME FANTASIA';
18    Emitente.Endereco.Fone              := '1532599600';
19    Emitente.Endereco.CEP               := '18270000';
20    Emitente.Endereco.Logradouro        := 'Rua Onze de Agosto';
21    Emitente.Endereco.Numero            := '1000';
22    Emitente.Endereco.Complemento       := '';
23    Emitente.Endereco.Bairro            := 'Centro';
24    Emitente.Endereco.Cidade.Codigo     := 3554003;
25    Emitente.Endereco.Cidade.Descricao := 'Tatuí';
26    Emitente.Endereco.UF                := 'SP';
27
28    Destinatario.CNPJCPF                    := '00000000000000';
29    Destinatario.Endereco.CEP               := '18270410';
30    Destinatario.Endereco.Logradouro        := 'Praça Anita Costa';
31    Destinatario.Endereco.Numero            := '0034';
32    Destinatario.Endereco.Complemento       := '';
33    Destinatario.Endereco.Bairro            := 'Centro';
34    Destinatario.Endereco.Cidade.Codigo     := 3554003;
35    Destinatario.Endereco.Cidade.Descricao := 'Tatuí';
36    Destinatario.Endereco.UF                := 'SP';
37    Destinatario.Endereco.Fone              := '1532599600';
38    Destinatario.IE                         := '000000000000';
39    Destinatario.NomeRazao                  := 'D.J. COM. E
                                                  LOCAÇÃO DE SOFTWARES
                                                  LTDA - ME';
40
41    with DadosProdutos.Add do
42    begin
43      CFOP          := 5101;
44      Codigo        := '67';
45      Descricao     := 'CEBOLA DESIDRATADA';
46      Quantidade    := 100;
47      Unidade       := 'KG';
48      ValorTotal    := 1000;
49      ValorUnitario := 10;
50      with Tributos do
51      begin
52        with ICMS do
53        begin
54          CST := '00';
55          ICMS00.Aliquota  := 18;
56          ICMS00.Valor     := 180;
57          ICMS00.ValorBase := 1000;
58        end;
59
60        with IPI do
61        begin
62          CST := '50';
63          FSituacaoTributaria := '50';
64          ClasseEnquadramento := '0';
65          CodigoSeloIPI := '0';
66          QuantidadeSelos := 1;
67          CodigoEnquadramento := '0';
68          ValorUnidade := 5;
69          CNPJProdutor := '00000000000000';;
70          ValorBase := 1000;
71          Aliquota := 5;
72          Valor := 50;
73        end;
74      end;
75    end;
76
77    with DadosProdutos.Add do
78    begin
79      CFOP          := 5101;
80      Codigo        := '68';
81      Descricao     := 'ALHO';
82      Quantidade    := 100;
83      Unidade       := 'KG';
84      ValorTotal    := 1000;
85      ValorUnitario := 10;
86      with Tributos do
87      begin
88        with ICMS do
89        begin
90          CST := '00';
91          ICMS00.Aliquota  := 18;
92          ICMS00.Valor     := 180;
93          ICMS00.ValorBase := 1000;
94        end;
95      end;
96    end;
97
98    with DadosProdutos.Add do
99    begin
100     CFOP          := 5101;
101     Codigo        := '69';
102     Descricao     := 'SAL';
103     Quantidade    := 100;
104     Unidade       := 'KG';
105     ValorTotal    := 1000;
106     ValorUnitario := 10;
107     with Tributos do
108     begin
109       with ICMS do
110       begin
111         CST := '00';
112         ICMS00.Aliquota  := 18;
113         ICMS00.Valor     := 180;
114         ICMS00.ValorBase := 1000;
115       end;
116     end;
117   end;
118
119   ValoresTotais.BaseICMS     := 3000;
120   ValoresTotais.ValorICMS    := 540;
121   ValoresTotais.ValorNota    := 3000;
122   ValoresTotais.ValorProduto := 3000;
123 end;
124
125 ACBrNFe1.Enviar(1);
126 memoResposta.Lines.Text := ACBrNFe1.WebServices.Retorno.RetWS;
```

A **Listagem 2** mostra como criar uma NF-e e enviá-la. A Linha 01 limpa as notas que possam estar na memória do componente. A Linha 03 adiciona uma nota para ser enviada, no caso de envio de notas em lote, o comando usado na Linha 03 poderá ser chamada diversas vezes, lembrando que o limite para envio é de 50 notas ou 500Kb. Os demais campos pelo próprio nome podemos identificar seu uso, apenas o campo Identificacao.Codigo que é utilizado na Linha 6 merece uma observação. Veja o que é falado sobre este campo no Manual de Integração: Código numérico que compõe a Chave de Acesso. Número aleatório gerado pelo emitente para cada NF-e para evitar acessos indevidos da NF-e. Então este código não deve ser igual o número da NF-e para evitar que outras pessoas além de quem criou e quem recebeu a NF-e consultem seu conteúdo no site da NF-e.

O comando utilizado na Linha 125 irá gerar, assinar, validar, transmitir, consultar se foi autorizado o uso da NF-e e em seguida imprimir o DANFE das NF-es autorizadas. O parâmetro 1 usado na Linha 125 se refere ao número do lote, que é um número seqüencial auto-incremental, de controle correspondente ao identificador único do lote enviado. A responsabilidade de gerar e controlar esse número é exclusiva do contribuinte.

Opcionalmente, você pode executar estes comandos separadamente, trocando o comando da Linha 125 pelos comandos mostrados na **Listagem 3**. Declare no início desta função uma variável de nome i e tipo integer.

Além de salvar o XML da NF-e, lembre-se de salvar ao menos a chave da NF-e, pois com ela que iremos executar as opções de consulta e cancelamento. Para obter a chave da NF-e acesse a propriedade ACBrNFe1.NotasFiscais.Items[i].XML.NFeChave onde i se refere ao número da NF-e dentro do lote enviado.

Listagem 3. Enviar NF-e com comandos separados

```
01  ACBrNFe1.NotasFiscais.GerarNFe;
02  ACBrNFe1.NotasFiscais.Assinar;
03  ACBrNFe1.NotasFiscais.Valida;
04
05  ACBrNFe1.WebServices.Envia(ALote);
06
07  for i:= 0 to ACBrNFe1.NotasFiscais.Count-1 do
08   begin
09     if ACBrNFe1.NotasFiscais.Items[i].XML.Confirmada then
10        ACBrNFe1.NotasFiscais.Items[i].XML.Imprimir;
11   end;
```

Nota

O serviço de recepção de NF-e é do tipo assíncrono, ou seja, a resposta não é imediata. Ao enviar um lote de NF-e, o WebService retorna um número de recibo informando que o Lote foi recebido com sucesso. Com este número é possível consultar o Status do Lote que pode retornar que ainda está em processamento ou o status da nota dizendo se foi ou não autorizada o uso ou se existe algum erro na mesma. O componente ACBrNFe, internamente ao enviar o lote já executa a consulta ao status do lote fazendo com que o serviço de envio pareça ser síncrono.

Consultar Status de uma NF-e

Após enviar uma NF-e, podemos consultar o status de uma NF-e de dois modos pelo ACBrNFe. Do primeiro modo, podemos usar apenas a chave da NF-e, que é gerada no envio da nota, ou carregar o arquivo XML da NF-e que foi enviada. Para o primeiro modo selecione o botão btnConsultar e no evento OnClick adicione o código mostrado na **Listagem 4**.

Listagem 4. Consultar NF-e com a chave de acesso

```
01 var
02   ChaveNFe : String;
03 begin
04   if not(InputQuery('Consultar NF-e', 'Chave da NF-e', ChaveNFe)) then
05      exit;
06
07   ACBrNFe1.WebServices.Consulta.NFeChave := ChaveNFe;
08   ACBrNFe1.WebServices.Consulta.Executar;
09   memoResposta.Lines.Text := ACBrNFe1.WebServices.Consulta.RetWS;
10 end;
```

É muito simples entendermos o código. Basta observar que estamos invocando um InputQuery para solicitar a Chave NF-e e em seguida passamos a informação ao Web Services e imprimimos o resultado em um Memo.

Para o segundo modo, adicione um componente OpenDialog da aba Dialogs e no evento OnClick do botão btnConsultar adicione o código mostrado na **Listagem 5**. Nesse caso estamos solicitando um arquivo XML para fazer a consulta. O resultado também é jogado no Memo.

Listagem 5. Consultar NF-e carregando o arquivo XML

```
01  OpenDialog1.Title := 'Selecione a NFE';
02  OpenDialog1.DefaultExt := '*-nfe.XML';
03  OpenDialog1.Filter := 'Arquivos NFE (*-nfe.XML)|*-nfe.
    XML|Arquivos XML
04    (*.XML)|*.XML|Todos os Arquivos (*.*)|*.*';
05  OpenDialog1.InitialDir := ACBrNFe1.Configuracoes.Geral.PathSalvar;
06  if OpenDialog1.Execute then
07  begin
08    ACBrNFe1.NotasFiscais.Clear;
09    ACBrNFe1.NotasFiscais.Add.XML.LoadFromFile(OpenDialog1.FileName);
10    ACBrNFe1.Consultar;
11    memoResposta.Lines.Text := ACBrNFe1.WebServices.Consulta.RetWS;
12  end;
```

Cancelar uma NF-e

Caso exista algum erro na NF-e ou por algum motivo o cliente desista da compra e a mercadoria ainda não tenha saído da empresa é possível cancelar a NF-e. Atualmente é possível efetuar o cancelamento de uma NF-e até 7 dias após a autorização do seu uso. Para cancelar uma NF-e é preciso a chave de acesso da NF-e e o número do Protocolo de Autorização. Caso você não tenha gravado o número do Protocolo de Autorização é possível obtê-lo utilizando o método mostrado na **Listagem 4** e acessar a propriedade ACBrNFe1.WebServices.Consulta.Protocolo. Para testar a funcionalidade de cancelamento do componente também temos duas opções. Podemos informar a chave de acesso da NF-e, o número do protocolo de autorização e a justificativa para cancelamento ou carregar o arquivo de uma NF-e e o componente irá consultar a NF-e para obter o número do protocolo para efetuar o cancelamento ou. Veremos o primeiro modo na **Listagem 6** e o segundo na **Listagem 7**. Atenção, o campo justificativa deve conter no mínimo 15 caracteres e não pode conter acentos ou caracteres especiais.

Listagem 6. Cancelar NF-e com chave de acesso e protocolo de autorização

```
01 var
02   ChaveNFe, Protocolo, Justificativa : String;
03 begin
04   if not(InputQuery('Cancelar NF-e', 'Chave da NF-e', ChaveNFe)) then
05      exit;
06   if not(InputQuery('Cancelar NF-e', 'Número do Protocolo',
      Protocolo)) then
07      exit;
08   if not(InputQuery('Cancelar NF-e', 'Justificativa',
      Justificativa)) then
09      exit;
10
11   ACBrNFe1.WebServices.Cancelamento.NFeChave       := ChaveNFe;
12   ACBrNFe1.WebServices.Cancelamento.Protocolo      := Protocolo;
13   ACBrNFe1.WebServices.Cancelamento.Justificativa  := Justificativa;
14   ACBrNFe1.WebServices.Cancelamento.Executar;
15   memoResposta.Lines.Text := ACBrNFe1.WebServices. Cancelamento.RetWS;
16
17 end;
```

Listagem 7. Cancelar NF-e carregando o arquivo XML

```
01 var
02   Justificativa : String;
03 begin
04   OpenDialog1.Title := 'Selecione a NFE';
05   OpenDialog1.DefaultExt := '*-nfe.XML';
06   OpenDialog1.Filter := 'Arquivos NFE (*-nfe.XML)|*-nfe.
      XML|Arquivos XML (*.XML)|*.XML|Todos os Arquivos (*.*)|*.*';
07   OpenDialog1.InitialDir := ACBrNFe1.Configuracoes.Geral.PathSalvar;
08   if OpenDialog1.Execute then
09   begin
10     ACBrNFe1.NotasFiscais.Clear;
11     ACBrNFe1.NotasFiscais.Add.XML.LoadFromFile(OpenDialog1.FileName);
12     if not(InputQuery('Cancelar NF-e', 'Justificativa',
        Justificativa)) then
13        exit;
14     ACBrNFe1.Cancelamento(Justificativa);
15     memoResposta.Lines.Text := ACBrNFe1.WebServices.Cancelamento.RetWS;
16   end;
17 end;
```

O cancelamento de NF-e é aceito quando o CStat de retorno está com o valor 101. A função de cancelamento retorna True quando o cancelamento é homologado em caso contrário é criado um exception mostrando a mensagem retornada pelo WebService.

Inutilizar Numeração

O serviço de inutilização de numeração deve ser usado quando a sequência numérica da NF-e for quebrada. Por exemplo: se a última nota emitida foi a de número 112 e por algum motivo a próxima nota foi a de número 120, deverá ser inutilizada a numeração de 113 até 119. Para inutilizar uma faixa de numeração, selecione o botão btnInutilizar e no evento OnClick adicione o código mostrado na **Listagem 8**.

O que fazemos em toda a sequência é solicitar ao usuário alguns dados básicos para a inutilização das notas. Pedimos: CNPJ do Emitente, Ano, Modelo, Serie, Número Inicial, Número Final e Justificativa. Por fim enviamos as informações ao Web Services e executamos usando o método Inutiliza. E claro atualizamos o nosso memo com a resposta.

Listagem 8. Inutilizar faixa de numeração da NF-e

```
01 var
02  CNPJ, Modelo, Serie, Ano, NumeroInicial, NumeroFinal,
     Justificativa : String;
03 begin
04  if not(InputQuery('Inutilizar Numeração', 'CNPJ do Emitente',CNPJ)) then
05     exit;
06  if not(InputQuery('Inutilizar Numeração', 'Ano',    Ano)) then
07     exit;
08  if not(InputQuery('Inutilizar Numeração', 'Modelo', Modelo)) then
09     exit;
10  if not(InputQuery('Inutilizar Numeração', 'Serie',  Serie)) then
11     exit;
12  if not(InputQuery('Inutilizar Numeração', 'Número Inicial',
     NumeroInicial)) then
13     exit;
14  if not(InputQuery('Inutilizar Numeração', 'Número Final',
     NumeroFinal)) then
15     exit;
16  if not(InputQuery('Inutilizar Numeração', 'Justificativa',
     Justificativa)) then
17     exit;
18
19  ACBrNFe1.WebServices.Inutiliza(CNPJ, Justificativa, StrToInt(Ano),
20    StrToInt(Modelo), StrToInt(Serie), StrToInt(NumeroInicial),
21    StrToInt(NumeroFinal));
22  memoResposta.Lines.Text :=  UTF8Encode(ACBrNFe1.WebServices.
     Inutilizacao.RetWS);
23 end;
```

Assim como no cancelamento, a justificativa deve conter no mínimo 15 caracteres, não pode conter acentos e caracteres especiais. O único modelo disponível atualmente é 55 e o ano deve ser informado apenas com 2 caracteres. Caso esteja cancelando apenas um número, informe no número inicial e número final o mesmo valor.

Validar NF-e

A validação feita pelo componente é realizada com base nos arquivos de Schemas fornecido pelo governo. Mesmo um arquivo sendo validado pelo componente, ao enviá-lo para o WebService em alguns casos é retornado a mensagem de Falha no Schema XML, este erro geralmente é causado por preenchimento incorreto de algum campo, pois a validação feita pelo componente apenas verifica se o conteúdo do arquivo está de acordo com o tipo de campo, por exemplo, o componente verifica se o campo numérico esta preenchido apenas com números, se está dentro do tamanho mínimo e máximo aceito pelo campo e se contém todos os campos exigidos. Ao enviar para o WebService outras validações são feitas e com esta mensagem genérica de erro fica difícil descobrir o que está errado. Caso receba esta mensagem, verifique se os valores monetários estão corretos, se a CFOP que está utilizando realmente existe e alguns campos dependem de tabelas divulgadas pelo governo, então verifique se o campo está preenchido de acordo com estas tabelas.

Para validar uma NF-e, também temos dois métodos. Após gerar uma NF-e pelo componente usando os comandos mostrados na **Listagem 2**, é possível usar o método ACBrNFe1.NotasFiscais.Valida ou carregando um arquivo XML. Para testar esta funcionalidade, selecione o botão btnValidar e no evento OnClick adicione o código mostrado na **Listagem 9**. Perceba que não há segredos, apenas limpamos as Notas Fiscais e carregamos um arquivo XML assim que o usuário o seleciona na caixa de diálogo. Em seguida Assinamos e Validamos usando os métodos correspondetes.

Listagem 9. Validar um NF-e carregando um arquivo XML

```
01  OpenDialog1.Title := 'Selecione a NFE';
02  OpenDialog1.DefaultExt := '*-nfe.XML';
03  OpenDialog1.Filter := 'Arquivos NFE (*-nfe.XML)|*-nfe.
     XML|Arquivos XML
04    (*.XML)|*.XML|Todos os Arquivos (*.*)|*.*';
05  OpenDialog1.InitialDir := ACBrNFe1.Configuracoes.Geral.PathSalvar;
06  if OpenDialog1.Execute then
07  begin
08    ACBrNFe1.NotasFiscais.Clear;
09    ACBrNFe1.NotasFiscais.Add.XML.LoadFromFile(OpenDialog1.FileName);
10    ACBrNFe1.NotasFiscais.Assinar;
11    ACBrNFe1.NotasFiscais.Valida;
12    ShowMessage('Nota Fiscal Eletrônica Valida');
13  end;
```

A procedure ACBrNFe1.NotasFiscais.Valida irá gerar um exception quando encontrar algum erro, caso contrário nenhuma mensagem será exibida.

Imprimir DANFE

Ao enviar uma NF-e com a função ACBrNFe1.Enviar, por padrão o componente já imprime automaticamente, mas podemos usar os comandos separadamente ou passar para a função Enviar além do número do lote um segundo parâmetro Imprimir informando False para que a impressão não seja feita automaticamente. Caso o DANFE não seja impresso no momento do envio, ou seja preciso imprimir outra cópia iremos utilizar o botão btnImprimir. Selecione o botão btnImprimir e no evento OnClick adicione o código mostrado na **Listagem 10**.

Listagem 10. Impressão do DANFE

```
01  OpenDialog1.Title := 'Selecione a NFE';
02  OpenDialog1.DefaultExt := '*-nfe.XML';
03  OpenDialog1.Filter := 'Arquivos NFE (*-nfe.XML)|*-nfe.XML|Arquivos XML
04    (*.XML)|*.XML|Todos os Arquivos (*.*)|*.*';
05  OpenDialog1.InitialDir := ACBrNFe1.Configuracoes.Geral.PathSalvar;
06  if OpenDialog1.Execute then
07  begin
08    ACBrNFe1.NotasFiscais.Clear;
09    ACBrNFe1.NotasFiscais.Add.XML.LoadFromFile(OpenDialog1.FileName);
10    ACBrNFe1.NotasFiscais.Imprimir;
11  end;
```

Simples, apenas chamamos o método Imprimir(). Além da impressão em papel o ACBrNFe oferece a opção de gerar o DANFE em PDF para que possa ser enviado por e-mail para o destinatário. Para gerar o DANFE em PDF, adicione um Button abaixo do botão btnImprimir e mude as propriedades Name para btnGerarPDF e Caption para Gerar PDF NF-e, no evento OnClick deste componente adicione o código mostrado na **Listagem 11**. Outra vez muito simples, pois existe um método específico para geração do PDF, o método: ImprimirPDF().

O arquivo será criado na pasta do programa com o nome NFe seguido da chave de acesso da NF-e.

Listagem 11. Gerar DANFE em PDF

```
01  OpenDialog1.Title := 'Selecione a NFE';
02  OpenDialog1.DefaultExt := '*-nfe.XML';
03  OpenDialog1.Filter := 'Arquivos NFE (*-nfe.XML)|*-nfe.XML|Arquivos XML
04    (*.XML)|*.XML|Todos os Arquivos (*.*)|*.*';
05  OpenDialog1.InitialDir := ACBrNFe1.Configuracoes.Geral.PathSalvar;
06  if OpenDialog1.Execute then
07  begin
08    ACBrNFe1.NotasFiscais.Clear;
09    ACBrNFe1.NotasFiscais.Add.XML.LoadFromFile(OpenDialog1.FileName);
10    ACBrNFe1.NotasFiscais.ImprimirPDF;
11    ShowMessage('Arquivo criado
12      em:'+ExtractFileDir(application.ExeName)+'\'+
13      ACBrNFe1.NotasFiscais.Items[0].XML.NFe.InfNFe.Id+'.pdf');
14  end;
```

Conclusão

Como podemos ver a adaptação do sistema para emissão de NF-e com o uso do ACBrNFe tornar-se uma tarefa relativamente simples. As maiores dúvidas na implementação da NF-e em seu sistema será com o preenchimento de alguns campos, para diminuir essas dúvidas, leia com atenção o Manual de Integração disponibilizado no portal da NF-e e mantenha contato com um contador, assim, o processo se tornará um pouco mais fácil.

Como principal desenvolvedor do componente ACBrNFe, recebo alguns emails perguntando se o componente é estável ou se alguém já usa em produção, posso dizer que o componente é estável, alguns clientes da empresa que trabalho já utilizam em produção e diversos usuários de diversos estados já utilizam o componente com sucesso. E por se tratar de um projeto com código aberto, caso você encontre algum problema, você pode corrigir e enviar sua colaboração para que o projeto se torne cada vez melhor.

Para mais informações sobre o Projeto ACBr conheça o site do projeto no endereço http://acbr.sf.net. Para suporte relativo ao componente, usamos como canal oficial o site FórumWeb no endereço www.forumweb.com.br/foruns/index.php?showforum=465. Além desses canais, caso tenha alguma sugestão para melhora dos componentes do ACBr ou a correção de algum bug utilize o endereço acbr.sourceforge.net/mantis/ ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos intermediários sobre Delphi Win 32 e Delphi for .NET

Padrão TCollection no Delphi

Implementando Coleções de Objetos com TCollection

Giuliano Prado de Morais Giglio
giulianopmg@yahoo.com.br
É desenvolvedor Delphi desde 1997, com ampla experiência em aplicações Win32. Graduado em Informática pela UFJF, com Especialização em Desenvolvimento de Aplicações para Web pelo CES de Juiz de Fora-MG, e Mestrado em Computação pela UFF/Niterói-RJ. Atualmente é professor universitário em diversas instituições, em cursos de Sistemas de Informação, e atua como consultor, pesquisador e desenvolvedor de aplicações Java, sobretudo na plataforma J2EE para Web, e J2ME, sendo especialista em aplicações Mobile.

Resumo do DevMan

Coleções de objetos são uma forma de agruparmos objetos de um mesmo tipo, a fim de facilitar a sua manipulação, através de reutilização de código em estrutura de dados disponíveis nas APIs das plataformas para este fim. Neste caso, teremos classes de coleção, ou simplesmente Collections, que proveem uma interface de métodos e propriedades, os quais permitem a manutenção dos objetos dentro da coleção de forma segura e consistente.

Neste artigo elucidaremos os conceitos e a importância do uso de coleções, e proporemos mostrar sua aplicação em Delphi, através da utilização das classes TCollection e TCollectionItem para este fim.

Uma coleção é um tipo de dado estruturado usado para agrupar, em uma única unidade, vários elementos homogêneos ou heterogêneos. Conceitualmente, na programação orientada a objetos, a coleção é uma estrutura de dados (um objeto) que pode armazenar referências a outros objetos, ou seja, é uma estrutura usada para armazenar, recuperar e fazer operações sobre dados de vários objetos.

Nesse artigo veremos

- O conceito de coleção de objetos;
- Vantagens da utilização de classes que implementam coleções;
- Implementação de coleções no Delphi, utilizando TCollection;
- Análise da solução de suas vantagens e desvantagens.

Qual a finalidade

Demonstrar a implementação do conceito de agregação de objetos, através da classe TCollection e TCollectionItem disponíveis na API do Delphi.

Quais situações utilizam esse recurso?

Manipular um conjunto de objetos em um sistema orientado a objetos, de forma a privilegiar a reutilização de códigos, tornando essa tarefa mais rápida, segura e consistente.

Em técnicas de mapeamento objeto-relacional, coleções estão intrinsecamente envolvidas, a fim de prover uma facilidade no trato dos objetos envolvidos entre o banco relacional e os programas orientados a objetos.

Normalmente, as coleções contêm referências a objetos que são inteiramente do mesmo tipo. As suas interfaces de estrutura declaram as operações a serem realizadas genericamente em vários tipos de coleções.

Com as coleções, os programadores utilizam estruturas de dados existentes, sem se preocupar com a maneira como elas são implementadas. Esse é um exemplo maravilhoso de reutilização de código. Os desenvolvedores podem codificar mais rápido e esperar um bom desempenho, com maximização da velocidade de execução e minimização do consumo de memória.

Como outras vantagens adicionais, podemos listar:

• Redução do esforço de programação, a partir do uso de estruturas de dados e algoritmos úteis, para diminuir a reescrita;

• Aumento da performance: como as várias implementações de cada interface são substituíveis, os programas podem ser facilmente refinados trocando-se as implementações;

• Reduz o esforço de aprendizado de API's, eliminando a necessidade de se aprender várias API's de coleções diferentes;

• Redução do esforço de projetar e implementar API's, eliminando a necessidade de se criar API's de coleções próprias;

• Promover o reuso de software, fornecendo uma interface padrão para coleções e algoritmos para manipulá-los.

Em particular nesse último item, normalmente os programadores implementam coleções de objetos através da simples implementação de Arrays. Desta forma, deveremos toda vez que estabelecer um novo Array de um tipo de objeto específico, implementar as suas funções de manipulação, tais como a inserção, remoção, buscas diversas, dentre outras. Temos assim, um processo de reescrita de código, dificultando a reutilização e a fácil implementação do conjunto de objetos.

Coleção de objetos, portanto, se caracteriza pela implementação do conceito de Agregação, um tipo de relacionamento um para muitos, sendo um padrão comumente utilizado em programação orientada a objetos. Agregação significa que uma classe tem uma lista de instâncias de outras classes, o que na verdade vem de encontro com a definição de coleções.

Podemos ter exemplos de agregação e como ela é usada para modelar um relacionamento de composição entre elementos do modelo, quando uma biblioteca contém Livros, ou dentro de uma empresa, quando Departamentos são constituídos de Funcionários. Para modelarmos isso, o (Departamento) agregado possui uma associação de agregação com suas partes integrantes (Funcionário).

Neste artigo mostraremos como se implementar coleções de objetos através das classes TCollection e TCollectionItem, fornecidas pela API do Delphi, bem como discutir as vantagens e desvantagens de sua utilização, se comparado com outras classes disponíveis, como TList, por exemplo.

Conhecendo TCollection

TCollections são também muito utilizados como parte do sistema de componentes do Delphi, oferecendo, portanto, suporte a serealização, atribuição e acesso a componentes agregantes (containers), incluindo a capacidade de notificá-los quando um objeto que está contido no mesmo for modificado. Isto pode ser interessante caso a necessidade seja a manipulação dos componentes VCL, mas pode ser custoso se desejarmos uma simples coleção de objetos. A hierarquia de TCollection pode ser vista na **Figura 1**.

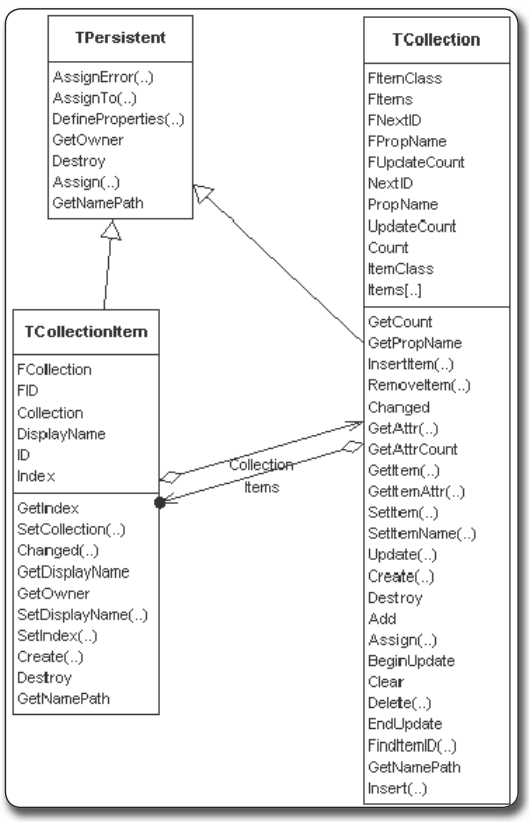


Figura 1. Hierarquia de TCollection

Podemos observar na **Figura 1** que a classe TCollection possui vários métodos e propriedades implementadas para a manipulação dos objetos que serão agregados. A agregação por TCollection é feita utilizando também a classe TCollectionItem. As duas classe herdam de TPersistent, a qual não abordaremos nesse texto.

Os membros da classe TCollection (propriedades e métodos) estão implementados para aceitar somente objetos TCollectionItem, ou seja, TCollection só poderá admitir coleções de TCollectionItem. Mas como poderíamos criar uma coleção de objetos, como por exemplo, produtos ou clientes? Faremos um exemplo explicativo para mostrar essa implementação.

Nota do DevMan

API (Application Programming Interface), ou interface de programação de aplicativos, é definida como um conjunto de funcionalidades, tais como procedures, métodos, classes, bibliotecas ou serviços disponibilizados pelos sistemas operacionais afim de prover recursos de interatividade e facilidade no desenvolvimento de softwares.

Implementando TCollection

Criaremos duas classes:

TProduto: classe representando um objeto produto;

TSetProduto: classe que conterá a coleção de produtos;

Como TSetProduto será nossa coleção, deveremos torná-la uma subclasse de TCollection. Dessa forma, TSetProduto estenderá todos os métodos de manipulação dos objetos, porém será necessário redefini-los.

Mencionamos anteriormente que TCollection só aceita agregar objetos do tipo TCollectionItem, logo, faremos com que TProduto seja subclasse de TCollectionItem, bem como a classe TSetProduto, como vimos na **Figura 2** seguir:

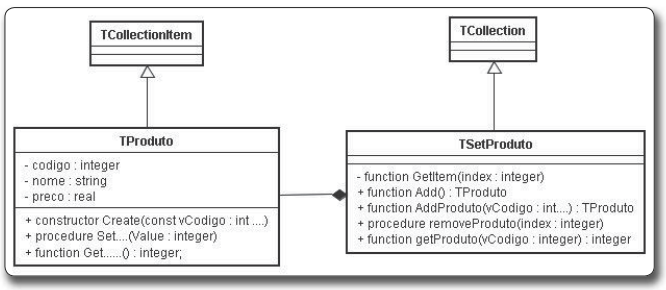


Figura 2. Estrutura hierárquica das classes implementadas

Nota do DevMan

Os modelos de classes feitos foram implementados pela ferramenta StarUml (http://staruml.sourceforge.net/en/index.php), sendo um projeto Open Source para o desenvolvimento rápido e flexível de diagramas dos mais diversos tipos, que utilizam o padrão UML/MDA (Model Driven Architecture - Modelo de Arquitetura Conduzida).

Seu desenvolvimento se restringe à plataforma Windows e, para atender às exigências de funcionalidades dos profissionais, a ferramenta fornece uma arquitetura simples e poderosa, na qual o usuário pode desenvolver os projetos voltados a linguagens de programação compatíveis (C++, Delphi, C#, VB). Podemos citar outros recursos fornecidos, tais como:

- Implementação de praticamente todos os diagramas da UML 2.0 atual (Classes, Casos de Uso, Atividade, Colaboração, Estado, Seqüência, Objetos, Componentes)
- Engenharia Reversa total e parcial, dependendo da linguagem.
- Geração de Documentação em formatos como PDF, HTML e de forma organizada.
- Facilidade de instalação e portabilidade entre as as diferentes versões da ferramenta (o que é uma restrição em outras ferramentas do mercado)
- Compreende todo o processo de modelagem da análise OO até a geração de código, nas linguagens supracitadas.

Podemos implementar a classe TProduto como na **Listagem 1**.

Definimos os atributos com acesso private e implementamos os seus métodos públicos get/set para acessá-los e modificá-los. Repare que a classe herda de TCollectionItem (Linha 02) para satisfazer a condição de agregação por TCollection.

A implementação dos métodos da Listagem 1 pode ser visto na **Listagem 2** a seguir. Nela temos o construtor create inicializando os atributos com os parâmetros passados, bem como a implementação dos métodos get/set para o atributo codigo, o que é extensivo aos demais, no caso, nome e preco.

Listagem 1. Interface da classe TProduto

```
01 type
02 TProduto = class(TCollectionItem)
03 private
04   codigo: Integer;
05   nome: String;
06   preco: Real;
07 public
08   constructor Create(const vCodigo: Integer; const vNome: String;
09                      const vPreco: Real);
10   procedure Setcodigo(const Value: Integer);
11   procedure Setnome(const Value: String);
12   procedure Setpreco(const Value: Real);
13
14   function Getcodigo: Integer;
15   function Getnome: String;
16   function Getpreco: Real;
17 end;
```

Listagem 2. Implementação dos métodos de TProduto

```
01 implementation
02
03  { TProduto }
04
05  constructor TProduto.Create(const vCodigo: Integer; const
      vNome: String;
06                              const vPreco: Real);
07  begin
08   codigo := vCodigo;
09   nome := vNome;
10   preco := vPreco;
11  end;
12
13  function TProduto.Getcodigo: Integer;
14  begin
15   Result := codigo;
16  end;
17
18  procedure TProduto.Setcodigo(const Value: Integer);
19  begin
20   codigo := Value;
21  end;
```

Devemos agora criar nossa classe TSetProduto baseada em TCollection, ou seja, tornando aquela, subclasse desta última (Linha 02) (**Listagem 3**):

Listagem 3: Interface da classe TProduto

```
01 type
02    TSetProduto = class(TCollection)
03 private
04   function GetItem(index: Integer): TProduto;
05 public
06   property Item[Index: Integer]: TProduto read GetItem;
07   function Add: TConta;
08   function AddProduto(const vCodigo: Integer; const vNome: String;
09                       const vPreco: Real): TProduto;
10   procedure removeConta(index: Integer);
11   function getProduto (index: Integer): Integer;
12 end;
```

Em um próximo passo, deveremos implementar os métodos para controlar os objetos dentro da coleção, sendo adicionar, remover e capturar um item, na verdade, um objeto. Os métodos criados substituirão os métodos da super-classe TCollection, e não serão estendidos, ou sobrescritos.

O motivo pelo qual não sobrescrevemos se deve ao fato de que os métodos estendidos retornam o mesmo tipo dos métodos da classe pai, que no caso da TCollection retorna TObject. Sendo assim, trocamos o tipo que esses métodos retornam de

TCollectionItem para TProduto, realizando o devido TypeCasting e garantindo a coleção de produtos desejada. Podemos observar isso nas Linhas 03 e 14 da **Listagem 4** a seguir:

Listagem 4. Implementação dos métodos de inclusão e remoção

```
01 function TSetProduto.Add: TProduto;
02 begin
03   Result := inherited Add as TProduto;
04 end;
05
06 procedure TSetProduto.removeProduto(index: Integer);
07 begin
08   inherited Delete(index);
09 end;
10
11 function TSetProduto.AddProduto(const vCodigo: Integer; const vNome:
12                          String; const vPreco: Real): TProduto;
13 begin
14   Result := inherited Add as TProduto;
15   Result.Setcodigo(vCodigo);
16   Result.Setnome(vNome);
17   Result.Setpreco(vPreco);
18 end;
```

O método Add, além de adicionar o item na coleção, retorna o próprio item adicionado. Alternativamente fizemos um método AddProduto no qual são passados por parâmetros os valores de um novo produto, os quais posteriormente poderão ser oriundos de um TForm, por exemplo.

No caso da remoção de um objeto, chamamos o método da super-classe Delete, passando o índice do elemento a ser excluído.

Neste exemplo, criaremos ainda uma propriedade Item de vetor com o método privado GetItem e definiremos como tipo de retorno TProduto. Este método será utilizado para capturar um objeto Produto específico da coleção, necessitando do índice do elemento, isto é, sua posição no conjunto, como mostrado a seguir:

```
function TSetProduto.GetItem(index: Integer): TProduto;
begin
  Result := inherited Items[index] as TProduto;
end;
```

É importante observar que esses métodos foram implementados com chamadas aos métodos da classe herdada (TCollection), já que estes possuem, na super-classe, a lógica da manipulação da coleção já implementada. Porém, estamos realizando o TypeCasting dos objetos para TProduto, tanto na recuperação, quanto na inclusão.

Para complementar a classe, adicionamos o método getProduto o qual faz uma busca diferenciada, através do código do produto. Como praticamente todos os métodos de TCollection necessitam do índice do elemento para manipulação, essa busca, implementada na **Listagem 5**, procura se o produto está na coleção e, caso esteja, retorna a sua posição.

Listagem 5. Busca de um objeto TProduto pelo seu código

```
01 function TSetProduto.getProduto(vCodigo: Integer): integer;
02 var i: integer;
03 begin
04   Result := -1;
05   for i := 0 to Count - 1  do
06   begin
07     if (vCodigo = GetItem(i).Getcodigo) then
08       Result := i;
09   end;
10 end;
```

Testando a coleção

Inicialmente criaremos um simples formulário, segundo a **Figura 3**. Os campos TEdit são nomeados como edCodigo, edNome, edPreco e os botões, btInserir, btBuscar, btRemover, btListar. Cada componente tratará do dado/operação indicado pelo seu nome. O restante (Memo,Labels, etc) deixaremos com os nomes default.

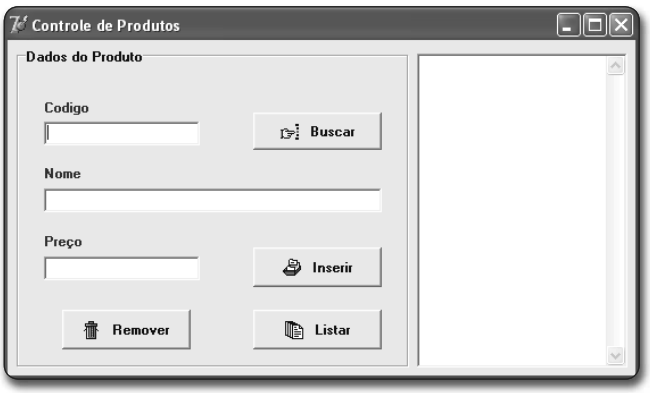


Figura 3. Formulário Principal de Controle de Produtos

Na seção implementation criaremos uma variável global para termos a TCollection implementada TSetProduto:

```
implementation
var
    produtos: TSetProduto;
```

Como se trata de um objeto, devemos instanciá-lo. Faremos essa operação no create do formulário.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  produtos := TSetProduto.Create(TProduto);
end;
```

No create do objeto TCollection devemos passar, por parâmetro, o nome da classe dos objetos que ela conterá, garantindo a integridade entre a coleção e os tipos de objetos que serão inseridos.

A inclusão dos produtos é feita pelo evento OnClick do botão btInserir, a seguir:

```
procedure TForm1.btInserirClick(Sender: Tbject);
begin
  produtos.AddProduto(strtoint(edCodigo.text),edNome.text,
                          strtofloat(edPreco.text));
end;
```

Podemos reparar que, no código do evento OnClick do btInserir, não há nenhum objeto TProduto, somente a coleção TSetProduto, e a partir dela, são instanciados os objetos TProduto, através do método AddProduto, o qual sabe que tipo de objeto criar, já que este foi informado à TCollection, no parâmetro de seu construtor. Deste modo, não deixa de ser uma certa economia de código e memória. O botão btBuscar tem a implementação da **Listagem 6**.

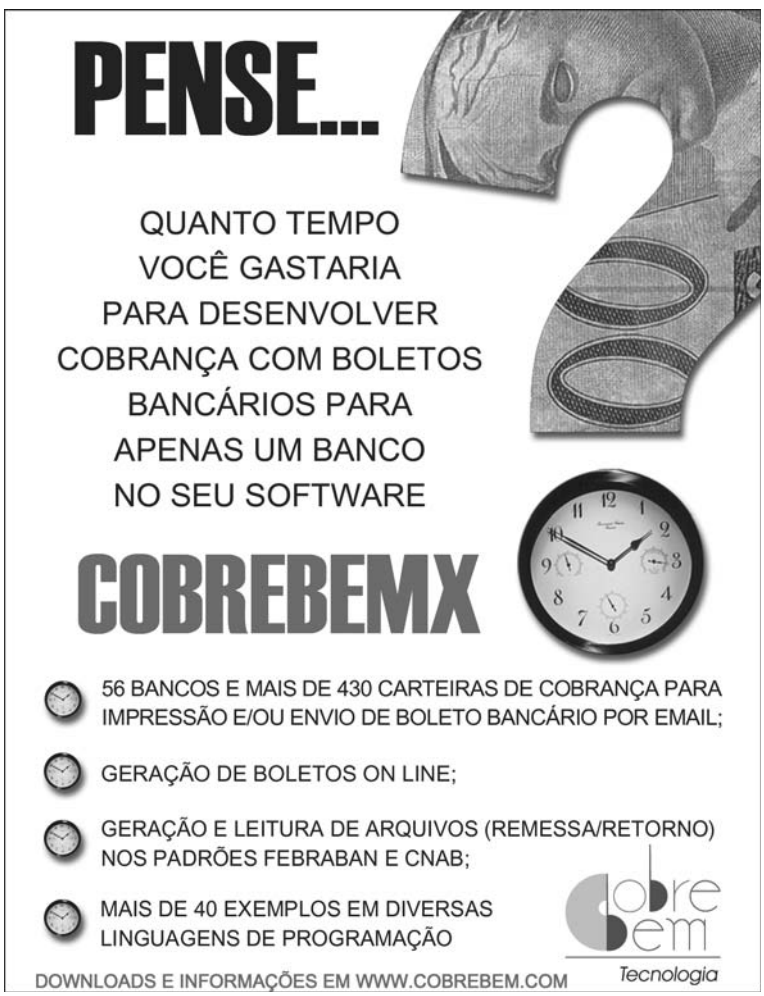
Listagem 6. Implementação do botão btBuscar

```
01 procedure TForm1.btBuscarClick(Sender: TObject);
02 var pos: integer;
03 begin
04   pos := contas.getProduto(strtoint(edCodigo.text));
05   if (pos <> -1) then
06    begin
07      edCodigo.text :=  inttostr(produtos.Item[pos].Getcodigo);
08      edNome.text := produtos.Item[pos].GetNome;
09      edPreco.text :=  floattostr(produtos.Item[pos].Getpreco);
10    end
11    else
12      Showmessage('Produto não encontrado!');
13 end;
```

Utilizamos o método getProduto para buscar um produto, através de um código preenchido no próprio campo edCodigo. Ao retornar o índice do objeto buscado, utilizamos este para acessar diretamente o objeto na coleção, através da propriedade Item. Logo, produtos.Item[pos] é um TProduto e, podemos então acessar seus atributos e métodos. Como os atributos são do tipo private, recuperamos seus valores através dos métodos Get.

Finalmente o botão btRemover (**Listagem 7**), que possui a seguinte implementação em seu evento OnClick:

Da mesma forma que o botão btBuscar, fazemos a busca pelo índice do objeto a ser removido e, caso ele esteja na coleção, passamos sua posição para o método removeProduto que faz a devida exclusão. Caso contrário, uma mensagem de erro é exibida ao usuário.



Listagem 7. Implementação do botão btRemover

```
01 procedure TForm1.btRemoverClick(Sender: TObject);
02 var pos: integer;
03 begin
04  pos := produtos.getProduto(strtoint(edCodigo.Text));
05  if pos <> -1 then
06   produtos.removeProduto(pos)
07  else
08   Showmessage('Produto não removido, por não ter sido encontrado!');
09 end;
```

Conclusão

O fato de podermos fazer uso de classes da API do Delphi, preparadas para a implementação de coleções de objetos, nos poupa tempo de codificação em relação a outras estruturas menos elementares (como Arrays, por exemplo). Além disso, temos maior segurança por estarmos utilizando estruturas robustas, bem como diminuir sensivelmente a reescrita e aumentar a possibilidade de reutilização com qualidade, privilegiando a produtividade no desenvolvimento de software.

A restrição imposta pela TCollection de se criar instâncias da sua coleção unicamente da classe que foi especificada em seu construtor, é uma forma de garantir a homogeneidade da coleção, ou seja, não há objetos diferentes ali inseridos a não ser do tipo que de fato desejamos. Desta forma, não só a inclusão, mas a recuperação dos objetos podem ser feitas de forma segura e sem a necessidade de castings adicionais, no ponto onde precisamos dos objetos, já que o mesmo é feito nos próprios métodos de manipulação da TCollection.

Uma desvantagem importante é que TCollection não suporta, portanto, classes múltiplas, mesmo que sejam derivadas do mesmo tipo base. Todo objeto na coleção precisa ser da mesma classe dos objetos agregados. Por exemplo, se quisermos ter uma coleção de TFuncionario, onde alguns são TGerente e TAtendente etc., teremos uma limitação. Uma vez que você registra que a coleção será em TFuncionario, somente este tipo será criado via método Add.

Em um segundo artigo, estaremos confrontando TCollection com outras técnicas disponíveis pelo Delphi para a implementação de coleções de objetos. Nele poderemos avaliar as vantagens e desvantagens sobre essas técnicas, sobretudo se compararmos com a TCollection demonstrada neste artigo, procurando oferecer ao leitor um panorama de soluções para a finalidade proposta, para que o mesmo tenha facilitada a sua escolha. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Chegou o Sistema de Créditos DevMedia

Agora o **conteúdo completo** do nosso site está **ao seu alcance!**

A partir de agora todas as **2000 vídeo-aulas** do site DevMedia podem ser compradas individualmente.

Economia - Você não precisa mais assinar oito produtos diferentes para ter acesso ao conteúdo completo do site!

Todas as vídeos que você sempre quis ver a partir **R$ 0,75!**

Saiba mais sobre o Sistema de Créditos!

DevMedia GROUP

www.devmedia.com.br/creditos

Nesta seção você encontra artigos intermediários sobre Delphi Win 32 e Delphi for .NET

Curso de Delphi Prism – Parte 1

Novidades da Linguagem – Variáveis Inline e Type Inference

Guinther Pauli
guinther@devmedia.com.br
Tem 21 anos de experiência na área de desenvolvimento e arquitetura de software, dos quais 14 dedicados ao Delphi e 9 ao .NET. Certificado Borland em Delphi 6, 7, 2005, 2006, Delphi para Web e Kylix, Certificado Microsoft MCP, MCAD, MCSD. NET, MCTS e MCPD. Editor geral da Revista .NET Magazine Brasil e professor de Pós-Graduação. Publicou mais de 150 artigos, 400 vídeo-aulas, palestrante em 5 edições da Borland Conference. Especialista em desenvolvimento Delphi, C#, .NET, ASP.NET, atuando também em projetos de migração orientando empresas e desenvolvedores na adotação de melhores práticas.

Resumo do DevMan

Neste artigo conheça um pouco da história do Delphi for .NET, Delphi Prism e dos IDEs do Delphi lançados até hoje. Começando o curso de Delphi Prism, tenha contato com as primeiras e mais importantes novidades da linguagem, como os recursos de variáveis inline, type inference e variáveis locais inicializadas e tipadas.

Um pouco de história da evolução do suporte ao .net no Delphi

O C# Builder foi a primeira ferramenta da Borland que suportou o .NET Framework. Porém, antes dele, o Delphi 7 até que trouxe um Preview que permitia o desenvolvimento de aplicações para .NET usando apenas um compilador, sem IDE. Já naquela época, por volta do ano de 2003, publicamos nossos primeiros artigos sobre a tecnologia. Desde então, acompanhamos a evolução do suporte ao desenvolvimento para .NET (veja meus artigos na edição 37). O C# Builder inaugurou um novo IDE da Borland, chamado BDS (Borland

Nesse artigo veremos

Lançado pela Embarcadero, empresa que adquiriu a CodeGear, o Delphi Prism é a nova forma de desenvolver aplicações para o .NET Framework e Mono utilizando o Object Pascal e o IDE do Visual Studio 2008. Nesta primeira parte do curso, são apresentados os novos recursos da linguagem utilizada pelo Delphi Prism, que são inúmeros se compararmos com o Delphi for .NET tradicional ou mesmo Delphi Win32.

Qual a finalidade

Comparar a sintaxe do antigo Pascal e Delphi tradicional com a moderna e poderosa linguagem suportada pelo Delphi Prism, conhecer os principais novos recursos da linguagem e vantagens.

Quais situações utilizam esse recurso?

O Delphi Prism abre uma porta incrível para os desenvolvedores Delphi, pois podem agora utilizar um IDE robusto, rápido e estável para construir aplicações para .NET, incluindo as novas tecnologias ASP.NET 3.5, ASP.NET AJAX, WPF, WCF, MONO etc. usando uma versão evoluída do Object Pascal, que mescla os melhores recursos do Delphi e do C# 3.0.

Developer Studio), que mais tarde, teve suporte ao Delphi na sua versão 2.0 (o Delphi for .NET). O Delphi 8 for the Microsoft .NET Framework (veja meu artigo na edição 48) foi o primeiro passo para que os desenvolvedores Delphi pudessem criar aplicações .NET usando tecnologias robustas, como ASP.NET, ADO.NET e Web Services, usando a nova plataforma da Microsoft. O Delphi 2005 uniu o Delphi Win32, Delphi for .NET e C# Builder em um único IDE (veja edição 57). Era o BDS 3.0. Depois disso, tivemos o suporte ao .NET 2.0 no RAD Studio 2007 e finalmente chegamos a 2009 com uma grande mudança de planos.

O BDS foi construído para incorporar uma série de linguagens em um único ambiente. É nele, por exemplo, que hoje desenvolvemos aplicações Delphi Win32, Delphi for .NET, C++ etc. Porém, para quem desenvolve para o .NET, sempre existiu um problema. Quando a Microsoft lança uma nova versão do .NET Framework, os desenvolvedores Delphi precisam esperar que um novo IDE seja lançado para suportar os novos recursos. Por exemplo, quando a Microsoft lançou o Visual Studio 2005 com suporte ao .NET Framework 2.0, os desenvolvedores Delphi somente 2 anos depois puderem contar com esse suporte, ainda limitado, no BDS.

Depois veio o .NET 3.0, 3.5 e agora o 4.0 (beta). A cada nova versão do framework, inúmeros recursos são lançados, novas tecnologias, novos controles, novos namespaces. Depois do .NET Framework 2.0, tivemos várias tecnologias lançadas disponíveis para os desenvolvedores que usam o Visual Studio, C# ou VB.NET, onde podemos destacar:

• LINQ, nova linguagem para consulta a dados, sejam eles provenientes de bancos de dados relacionais, documentos XML ou mesmo objetos;

• WPF – Windows Presentation Foundation, nova tecnologia para o desenvolvimento de interfaces gráficas (a grosso modo, imagine isso como um DFM em estrutura de markup XML);

• WCF – Windows Communication Foundation, que unifica em uma única tecnologia o desenvolvimento de aplicações distribuídas;

• WF – Windows Workflow Foundation, para a criação de estruturas de Workflow;

• ASP.NET AJAX, para o desenvolvimento de interfaces de alto desempenho para Web;
Silverlight (o concorrente do Flash) para aplicações Web ricas e inteligentes com experiência de usuário superior;

• ASP.NET MVC, para o desenvolvimento de aplicações Web usando o poderoso padrão MVC (Model – View – Controller);

• ADO.NET Entity Framework, o ECO da Microsoft, concorrente do NHibernate, para o desenvolvimento de aplicações usando um framework de persistência e mapeamento objeto relacional;

• ADO.NET Data Services - uma biblioteca do ADO.NET que tem o objetivo de disponibilizar o acesso, e até a alteração de dados através de serviços Web;

• Windows Azure – a plataforma para Cloud Computing (computação na nuvem) da Microsoft;

• C# 3.0 e 4.0 – a principal linguagem do .NET recebe dezenas de novidades a cada ano, estando em evolução constante;

Isso para citar por cima o que está em alta no desenvolvimento com .NET, muita coisa existe além disso, muitas coisas ainda estão em beta ou CTP.

Nota do DevMan

Foi anunciado em maio de 2009, a nova versão do Visual Studio, a 2010, ainda em beta. É provável que até o final do ano a versão final do produto já esteja disponível, juntamente com o .NET 4.0, que trará inúmeras novidades, como linguagens dinâmicas (DLR), computação paralela, Desenvolvimento Orientado à Testes (TDD), Cloud Computing, Team System, mais novidades no ASP.NET AJAX, Workflow, ADO.NET, CLR etc. Com o Delphi Prism, é muito provável que logo após esse lançamento já seja possível utilizar todos esses novos recursos usando Object Pascal. Aliás, é provável que em breve já tenhamos um beta do Prism para o VS2010.

O grande Q da questão é que os desenvolvedores Delphi sempre estavam atrás no que diz respeito ao desenvolvimento com o .NET, pois dependiam do lançamento de um novo IDE que suporte uma nova versão do framework. Antes do Prism, por exemplo, já se completariam 4 anos que não tínhamos como usar novas versões do .NET com Delphi.

O Delphi Prism

É fato que muitos desenvolvedores Delphi, ao decidirem adotar o .NET Framework, principalmente para soluções Web com ASP.NET, acabaram por adotar o Visual Studio com o C#. Pensando nisso, há alguns anos atrás, a empresa Remote Objects, então parceira da Borland, criou o projeto Chrome. Assim como o BDS, o Visual Studio também é um IDE que permite a incorporação de novas linguagens em seu IDE. Então, a equipe da RemObjects criou uma nova linguagem, uma espécie de evolução do Object Pascal, importando ainda muitas coisas interessantes que o C# e suas novas versões oferecem. Essa linguagem não usa o BDS como IDE, mas o próprio Visual Studio. O Chrome então é o melhor de dois mundos: uma linguagem clássica e robusta, um Object Pascal turbinado (que pessoalmente eu prefiro chamar de D#) dentro de um IDE já consolidado, estável e produtivo, o Visual Studio (atualmente na versão 2008). O Chrome, mais tarde, veio a se chamar Oxygene, que foi então comprado pela Embarcadero, empresa que adquiriu a CodeGear, levando o intuitivo nome de Delphi Prism.

Em suma, podemos dizer então que o Delphi Prism é a tecnologia que nos permite desenvolver usando uma nova linguagem que é uma mescla do que existe de melhor no Object Pascal, Delphi Language e C# 3.0, dentro do IDE do Visual Studio, com suporte ao que existe de mais novo (e velho, desde que seja .NET) no mundo Microsoft (todos aqueles que citei anteriormente, incluindo WPF, LINQ e ASP.NET AJAX).

Na edição 105, tivemos aqui na ClubeDelphi uma introdução ao desenvolvimento de aplicações com o Delphi Prism. O objetivo desta série que agora iniciamos é mostrar o Prism em sua totalidade, começando pelas inúmeras vantagens da linguagem.

A linguagem do Delphi Prism

Desde o Turbo Pascal 1.0, muitas coisas se tornaram imutáveis e regras prevaleceram para a eternidade. Uma linguagem fortemente tipada, com uma sintaxe clara, orientada a objetos (desde o Turbo Pascal 5.5), fizeram dela muito popular e muito utilizada em escolas e universidades, principalmente para o ensino de lógica de programação e algoritmos. O Delphi 1 trouxe o Pascal para o mundo corporativo. Com o Delphi Prism, muitas exigências rígidas do Pascal se tornaram mais flexíveis, principalmente para incorporar na linguagem recursos presentes no C# 3.0. Então, gostaria de adverti-lo, que antes de mergulharmos a fundo na linguagem Delphi Prism, que você abra sua mente. Para começar, analise o seguinte código, escrito em Delphi Prism:

```
method _Default.Delphi_Prism_Instanciando_Guinther();
begin
    var Eu := new class (Nome := "Guinther", Idade := 31);
end;
```

Não, eu não declarei nenhum tipo, o var está dentro do bloco de código, a pessoa instanciada não é uma classe declarada, new é tão novo quanto o seu próprio nome sugere, o construtor é estendido, as propriedades do objeto são inicializadas Inline (não é overload não!), e o método não é procedure nem function, é method mesmo. Isso compila sem problemas. O objetivo desde exemplo é único: mostrar a você que a linguagem Delphi deu um grande salto, e novamente, para que você se liberte, abra sua mente, esteja pronto para ir além. No decorrer deste curso, você aprenderá em detalhes as sete novidades que mostrei no único código anterior, além de muitas outras (sim, agora também é possível usar aspas duplas no lugar de aspas simples).

O objeto desta série é ensinar, por completo, a você desenvolver com o novo Delphi Prism. Nossa jornada vai começar pelas novidades da linguagem. Sempre que aplicável, vou comparar o novo código com o que fazíamos no Delphi Win32. Depois, vamos fazer um tour pelo IDE, comparando ao Delphi 7 e ao BDS (em futuras edições da revista). Na sequência, vamos criar aplicações robustas para Web, com ASP.NET, vamos desenvolver aplicações com AJAX, WPF, LINQ, e tudo o que for suportado, tudo isso ao longo do curso. Esteja pronto para um novo mundo, uma nova evolução, um novo conceito na forma de criar aplicações com Delphi e Object Pascal (ou como eu prefiro dizer, D#).

Nota do DevMan

Onde estão a function e procedure? Essas são palavras herdadas do Pascal estruturado, quando ainda se chamavam "rotinas". Na orientação a objetos não existem rotinas, mas sim métodos, que nada mais são que procedures ou functions dentro de classes. Então, por que não chamar isso de method? É simples usar no Prism, se for procedure, declare normalmente usando a nova palavra reservada method. Se quiser criar uma função, faça o mesmo, e apenas coloque um ":" no final e o tipo de retorno:

```
// procedure
method _Default.TesteProdecedure();
// function
method _Default.TesteFunction(): Integer;
```

Como criar os exemplos deste curso

Vamos então começar pelas novidades da linguagem, uma a uma. Para criar os código que serão apresentados a seguir, você pode criar qualquer tipo de aplicação .NET, seja ela Web ASP.NET, Windows Forms, Web Service, Console etc. Aqui vou utilizar sempre aplicações ASP.NET, mas repare que o tipo da interface não é importante, o foco aqui é a LINGUAGEM. Para criar suas aplicações Web com Delphi Prism, no Visual Studio, sempre siga os seguintes passos (veja **Figura 1**):

- Clique em File>New Web Site;
- Na janela que aparece, clique em ASP.NET Web Site;
- Escolhe a versão do .NET Framework que deseja utilizar;
- Escolha File System para o Location;
- Dê um nome ao diretório onde será criado o Web Site.

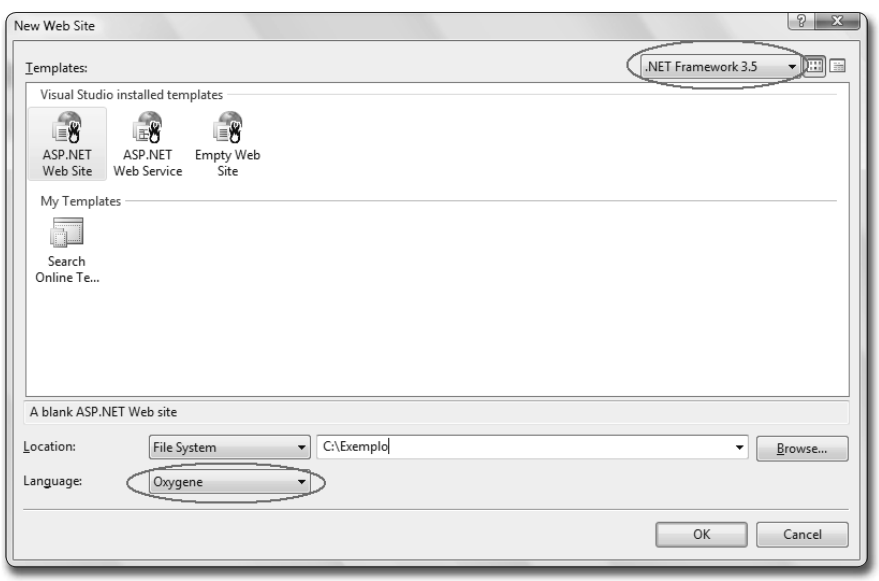


Figura 1. Criando um ASP.NET Web Site

Clicando em Ok, um novo Web Site será criado. Desde o Visual Studio 2005, não é preciso hospedar o Web Site no servidor IIS, pois o IDE usa um servidor local para hospedar e rodar a aplicação em teste. Para abrir um Web Site, basta dar o Open e escolher o diretório. Para rodar um exemplo, basta abrir e executar, como uma aplicação Delphi tradicional. Isso é devido à opção File System. Se já quiser sair usando o IIS, escolha HTTP. Oxygene indica que queremos usar Delphi Prism como linguagem para o ASP.NET Web Site. Veja que no momento da criação do Web Site, podemos escolher qual linguagem vamos utilizar, C#, VB.NET ou Delphi Prism (além de outra que você tenha instalado). Também é possível escolher qual versão do framework queremos utilizar (esse é um novo recurso do VS2008).

Após o Web Site ser criado, localize a janela Solution Explorer (lembra do Project Manager?) e abra o arquivo Default.aspx.pas. Basicamente ele possui a estrutura mostrada na **Listagem 1**. Novamente, como não vamos desenvolver nenhuma aplicação aqui, e sim testar as novidades da linguagem, vamos inserir os códigos em um método qualquer, no Page_Load por exemplo. Vamos então, uma a uma, conhecer as novidades da linguagem.

Listagem 1. Estrutura do arquivo pas (code-behind) do ASP.NET Web Site

```
namespace;

interface

uses
  System,
  System.Data,
  System.Configuration,
  System.Web,
  System.Web.Security,
  System.Web.UI,
  System.Web.UI.WebControls,
  System.Web.UI.WebControls.WebParts,
  System.Web.UI.HtmlControls;

type
  _Default = public partial class(System.Web.UI.Page)
  protected
    method Page_Load(sender: Object; e: EventArgs);
  end;

implementation

method _Default.Page_Load(sender: Object; e: EventArgs);
begin

end;

end.
```

Variáveis Inline

O Pascal sempre foi uma linguagem fortemente tipada. Algumas exceções existem a esta regra, como os Variants e OleVariants criados no Delphi 2, ou mesmo tipos Pointer que apontam para qualquer coisa. Além disso, qualquer programador Delphi ou Pascal aprende no primeiro dia de aula que variáveis devem ser sempre declaradas antes de serem utilizadas, seja na sessão var de um procedure/function, dentro da estrutura de uma classe, globalmente a uma unit etc. Veja um exemplo na **Listagem 2**.

Listagem 2. Pascal tradicional, com sessão var

```
procedure TForm1.FormCreate(Sender: TObject);
var
  s: string;
  i: integer;
  b: boolean;
  n: double;
begin
  s := 'Guinther';
  i := 31;
  b := true;
  n := 19.28;
  // código que usa as variáveis
  ShowMessage(s);
  ShowMessage(IntToStr(i));
  ShowMessage(BoolToStr(b,true));
  ShowMessage(FloatToStr(n));
end;
```

No C# não existe o conceito de sessão var. Você vai codificando seu método, e no momento que necessitar de uma variável, seja lá de que tipo for, é possível declará-la no meio do código, já a inicializando, e a seguir utilizá-la normalmente. Veja um exemplo na **Listagem 3**. Repare que o tipo da variável é declarado antes do seu nome, ao contrário do Pascal.

Listagem 3. Código C# com variáveis Inline

```
protected void Page_Load(object sender, EventArgs e)
{
    string s = "Guinther";
    Response.Write(s);
    int i = 31;
    Response.Write(i.ToString());
    bool b = true;
    Response.Write(b.ToString());
    double n = 19.28;
    Response.Write(n.ToString());
}
```

Nota

Ao longo deste curso, vou fazer várias referências ao C#. Mesmo usando o Delphi Prism, é importantíssimo que você conheça a semântica da linguagem C#, pois ao desenvolver para .NET, toda documentação, livro, artigo e outros recursos que você for utilizar, provavelmente será escrito em C# ou VB.NET. Mas não se preocupe, é muito simples identificar a diferença entre um {} e begin end!

Agora, em Delphi Prism, podemos fazer o mesmo. Observe na **Listagem 4** o código equivalente ao Delphi Win32 da **Listagem 2** e ao código C# da **Listagem 3**. Se quiser compilar a aplicação para testar o código, basta dar um Build. Se quiser visualizar a saída no browser, rode com F5 ou Ctrl+F5. Observe a semelhança do código Prism para o C#.

Listagem 4. Código Delphi Prism com variáveis Inline

```
method _Default.Page_Load(sender: Object; e: EventArgs);
begin
    var s: string := "Guinther";
    Response.Write(s);
    var i: integer := 31;
    Response.Write(i.ToString());
    var b: Boolean := true;
    Response.Write(b.ToString());
    var n: Double := 19.28;
    Response.Write(n.ToString());
end;
```

Type Inference

Você já reparou, claro, que no Delphi sempre declaramos o tipo e, quando ele for um objeto, o instanciamos com Create chamando o construtor da classe de mesmo nome. Na **Listagem 5**, declaramos um TClientDataSet, e a seguir, instanciamos um TClientDataSet. Mas veja bem, para que eu preciso declarar a classe como sendo TClientDataSet, se depois no momento da criação eu atribuo um objeto do mesmo tipo TClientDataSet à variável?

Listagem 5. Declarando e instanciando objetos no Delphi

```
procedure TForm1.Teste_Type_Inference();
var
  Cds: TClientDataSet;
begin
  Cds := TClientDataSet.Create(nil);
end;
```

Isso tem uma explicação. Em alguns casos, uma variável não é declarada do mesmo tipo que vai ser instanciada, mas sim de um tipo ancestral. Assim, é possível aplicar técnicas de orientação a objeto como polimorfismo, quando chamamos um método virtual (é por esse motivo, por exemplo, que o Quick Report aponta para um TDataSet e até hoje pode suportar inúmeros engines de acesso a dados). Veja um exemplo na **Listagem 6**. Aqui declaramos uma variável do tipo TDataSet, classe base para muitas outras classes de acesso a dados na VCL. Porém, veja na implementação que Ds pode receber tanto um TQuery quanto um TClientDataSet, pois ambos descendem de TDataSet. Quando chamamos Ds.Post, o compilador vai analisar a assinatura do método na classe declarada (TDataSet), e vai detectar que ele é virtual. Com isso, o verdadeiro Post que será chamado não é o da classe TDataSet, mas de TClientDataSet, que sobrescreve o Post com override, configurando uma chamada polimórfica.

Listagem 6. Polimorfismo

```
procedure TForm1.Polimorfismo();
var
  Ds: TDataSet;
  b: boolean;
begin
  b:= false; //ou outra expressão booleana
  if b then
    Ds := TQuery.Create(nil)
  else
    Ds := TClientDataSet.Create(nil);
  // aqui configura DS
  // chama método polimórfico
  Ds.Post();
end;
```

E no Prism? Bem, é possível fazer tudo isso sem problemas no Prism, usando objetos da FCL ao invés da VCL claro. Porém, aqui temos uma novidade. Se não formos usar polimorfismo, ou seja, o tipo que vamos instanciar é o mesmo que vai ser declarado (Inline ou na sessão var), podemos omitir o tipo. Veja como seria isso analisando o código da **Listagem 7**. No primeiro código, um trecho normal, usando apenas variáveis Inline. Declaramos o objeto, do tipo DataSet, e o instanciamos a partir do construtor da classe DataSet (veja que a nova palavra-chave para instanciar objetos é NEW, e não mais Create). No segundo código, o tipo foi omitido, ou seja, foi declarado sem tipo. O compilador do Delphi Prism, a partir do valor que for jogado dentro da variável Ds2, determina o tipo do objeto. Na sequência, outros exemplos de como usar o recurso. Isso é chamado de Type Inference (não são Variants não, são variáveis fortemente tipadas).

Listagem 7. Type Inference

```
method _Default.Page_Load(sender: Object; e: EventArgs);
begin
  // Declarando o tipo
  var Ds1: DataSet := new DataSet();
  // Type Inference, sem declarar o tipo
  var Ds2 := new DataSet();
  // Outros exemplos de Type Inference
  var i := 31;
  var s := "Guinther";
end;
```

Vamos fazer uma comprovação. Na última linha do código da **Listagem 7**, antes do end, escreva o seguinte:

```
Response.Write(s.GetType().ToString());
```

Como você viu, a variável s foi declarada sem tipo. Então, usamos Reflection (o RTTI – Runtime Type Information, do .NET) para verificar qual foi o tipo de objeto utilizado pelo compilador para declarar s nos bastidores. A saída vai ser:

```
System.String
```

Obviamente, esse recurso também existe no C#. O equivalente seria:

```
var s = "Guinther";
```

Que para o compilador, equivale a:

```
string s = "Guinther";
```

Variáveis Locais Tipadas e Inicializadas

No Delphi Win32, é possível declarar uma variável e automaticamente inicializá-la, como mostrado a seguir:

```
var
  s: string = 'Guinther';
```

Mas veja que isso só funciona para variáveis GLOBAIS (aliás, se você ainda usa variáveis globais, está mais do que na hora de rever seus conceitos como desenvolvedor). Você não pode, por exemplo, tentar inicializar uma variável declarada dentro de um escopo de um método:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  s: string = 'Guinther'; // Erro
```

O compilador, nesse caso, vai emitir a seguinte mensagem de erro de compilação:

```
[Error] Unit1.pas(29): Cannot initialize local variables
```

Sinceramente, eu não sei o porquê dessa limitação. Ainda mais que, você pode usar POG (Programação Orientada a Gambiarra) para driblar o compilador do Delphi Win32 e fazer como na **Listagem 8**. Aqui declaramos não uma variável local inicializada (o que geraria um erro), mas uma constante local, tipada e inicializada (sim, isso é possível). Constantes, ao contrário do que o próprio nome sugere, podem ser alteradas em runtime, desde que você ative a diretiva de compilação {$J+}. Pronto, acabamos de enganar o compilador e simulamos uma variável local tipada inicializada.

Listagem 8. POG para obter variáveis locais tipadas inicializadas

```
procedure TForm1.FormCreate(Sender: TObject);
{$J+}
const
  s: string = 'Guinther';
begin
  s := s + ' Pauli';
```

No Prism, não precisamos de POG, é completamente válido escrever como na **Listagem 9**.

Listagem 9. No Prism é possível inicializar variáveis locais

```
method _Default.Page_Load(sender: Object; e: EventArgs);
var
  s: String := "Guinther";
begin
  s := s + " Pauli";
  Response.Write(s);
end;
```

Além disso, é perfeitamente possível usar Type Inference para obter o mesmo resultado, ou seja, inicializar uma variável omitindo seu tipo, como na **Listagem 10**.

Listagem 10. Variável local inicializada com Type Inference

```
method _Default.Page_Load(sender: Object; e: EventArgs);
var
  s := "Guinther";
begin
  s := s + " Pauli";
  Response.Write(s);
end;
```

Conclusão

Nesta primeira parte do nosso curso, vimos o que considero serem as mudanças mais marcantes da nova linguagem instituída pelo Delphi Prism. Provavelmente, a possibilidade de declarar variáveis Inline deve ter mexido com a cabeça de muita gente. Alguns podem pensar Ah, eu prefiro usar a sessão var, o meu código fica mais organizado.... Mas veja que isso foram nossos antigos professores que 10 ou 20 anos atrás nos ensinaram. Se olhar bem, a possibilidade que essa nova abordagem nos dá, torna nosso código muito mais flexível, além de ser mais rápido de codificar. Você não precisa ir até a sessão var, declarar, voltar ao código e continuar o seu trabalho (até que o Refactoring pode ajudar nesse caso). Enfim, mesmo que não se sinta a vontade, ainda dá para usar o método antigo. Type Inference é outro excelente recurso, como pudemos comprovar. E finalmente, para se adequar ainda mais aos padrões do .NET Framework, muita coisa se tornou mais parecida com as outras linguagens do .NET ou mesmo outras linguagens famosas. Por exemplo, instanciamos objetos agora com o NEW.

Esses são os primeiros recursos que provavelmente causarão mais impacto no seu código e principalmente, na maneira de você pensar (para aqueles que não vão continuar fiéis ao Sr. Niklaus Wirth). Existe muito, mas muito a se falar apenas sobre a linguagem do Delphi Prism, assuntos que vamos abordar nas próximas edições em detalhes. Depois que conhecermos a sintaxe do Prism, vamos começar a utilizar as novas e poderosas tecnologias disponíveis no .NET Framework, como LINQ, WPF, ASP.NET AJAX, ADO.NET Entity Framework, MVC, Data Services, Azure e muito mais.

Bem vindo a uma nova era e até a próxima edição. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos para iniciantes na linguagem Delphi

ReportBuilder

Montando relatórios com o ReportBuilder

Thales Eduardo Barros de Oliveira
tebosoftware@gmail.com
É formado em Tecnologia em Telecomunicações pela Universidade de Franca (UNIFRAN), trabalha com Delphi desde 2002 e atua no setor de desenvolvimento de software para a indústria.

Resumo do DevMan

Dizer que praticamente 80% de um sistema é feito de relatórios não é nenhum exagero. Para que criamos um sistema com alimentação de dados se não podemos listar ou imprimir relatórios do que está no banco de dados? É assim que grande parte das pessoas entre clientes e desenvolvedores pensa. Realmente, uma das partes principais em um sistema são os relatórios.

O ReportBuilder é uma ferramenta de desenvolvimento de relatórios que consegue auxiliar, de maneira fácil e intuitiva, o montagem de relatórios desde os mais simples, com uma pequena listagem de clientes, até relatórios com agrupamentos, sub-relatórios ou até mesmo gráficos.

Nesse artigo veremos

•Principais componentes e funcionalidades do ReportBuilder;

• Criação de relatório simples com quebra por grupos e sumários;

• Principais conceitos no desenvolvimento de relatórios.

Qual a finalidade

Com o ReportBuilder temos como gerar desde simples listagens até relatórios extremamente complexos.

Quais situações utilizam esse recurso?

Podemos gerar relatórios para todo e qualquer tipo de software que venhamos a desenvolver.

Olá pessoal, neste meu primeiro artigo, gostaria de abordar uma das tarefas mais importantes no cotidiano de um sistema: o desenvolvimento de relatórios. Com certeza uma das etapas mais importantes no desenvolvimento de sistemas é poder dar a oportunidade ao usuário final de consultar e imprimir aquilo que foi incluído no banco de dados. É muito comum a apresentação de relatórios a determinados membros em uma empresa, por isso a necessidade sempre grande de se ter esse tipo de recursos no software. Imagine por exemplo um sistema de contas a pagar e receber que não nos permitir imprimir o fluxo de caixa da empresa. Praticamente o sistema torna-se inútil.

Existem milhares de editores de relatório

que podem ser ou não integrados ao Delphi para que possamos fazer tais desenvolvimentos. Um deles é o ReportBuilder, da empresa Digital Metaphors como irei explicar mais adiante.

Para isso vou demonstrar a utilização dessa ferramenta. Como dito anteriormente o ReportBuilder é um gerador de relatórios fornecido pela empresa Digital Metaphors e encontra-se na versão 11.04.

Nota

Nesse artigo utilizaremos a versão 10.06 juntamente com o RAD Studio 2007, porém esse exemplo pode ser desenvolvido a partir da versão 5 do Delphi.

Com esta ferramenta é possível desenvolver relatórios bastante complexos, como poderemos ver. No site do fabricante, em http://www.digital-metaphors.com/, existe uma versão demonstrativa para download. Não entrarei em detalhes quanto a instalação dos componentes por não fazer parte do escopo desse artigo. Porém, é muito simples instalá-los. Basta seguir o Wizard de instalação que não há nenhuma configuração em especial.

Neste primeiro exemplo, montaremos um relatório mais simples, que consiste em uma relação de pedidos, simples com agrupamento e resumo por pedido e um resumo geral de vendas.

Montando a base de dados do relatório

Bom para começarmos, criaremos uma pasta para guardar o nosso exemplo chamada Proj_ReportBuilder. Vamos precisar de uma fonte de dados para este exemplo e para isso, utilizaremos o banco de dados Postgres, mas você pode utilizar o banco de dados que preferir. Vamos ver rapidamente com fazer o download e instalação do PostGreSQL para que possamos prosseguirmos com o artigo.

Acesse o site do PostGreSQL através do link http://www.postgresql.org.br/ e clique em Download. Na tela existem todas as informações necessárias para download da versão mais apropriada ao nosso caso. Faremos então a instalação em um sistema Windows, portanto acesse o link Windows mais ao centro do site. Em seguida clique na opção Download the one click installer e aguarde o carregamento da página. Agora basta clicar no botão Windows e salvar o arquivo em um diretório de sua preferência. A versão que instalaremos é a 8.3.1-1.

Feito o download, execute o arquivo postgresql-8.3.7-1-windows.exe para que possamos iniciar a instalação. Não entrarei em detalhes quanto a instalação, apenas siga os procedimentos do próprio instalador. Um software para manutenção do banco será instalado no grupo Programas no botão Iniciar: o pgAdminIII.

Abra o pgAdminIII e crie um novo banco de dados chamado ReportBuilder, como mostro na **Figura 1**. Com o banco criado, execute o script da **Listagem 1** para que sejam criados as tabelas Pedido e Item, e juntamente inseridas algumas informações para nosso exemplo. Faça o download do script completo portal DevMedia, pois parte desse script foi omitido para poupar espaço na revista.

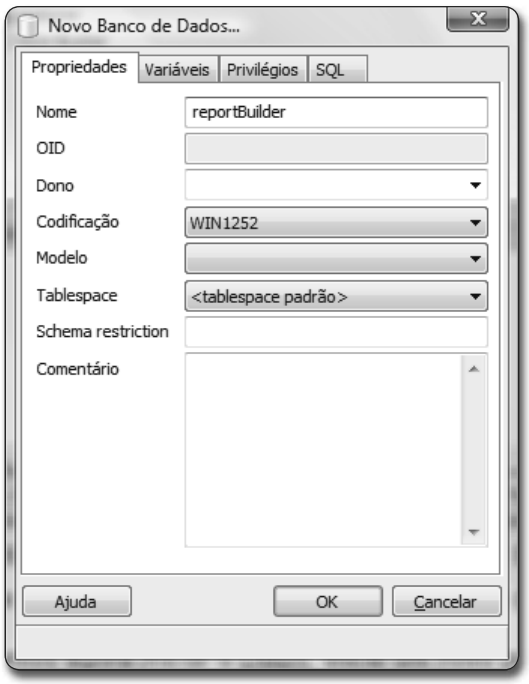


Figura 1. Criando um novo banco de dados

Listagem 1. Criação e inclusão de registros no banco de dados

```
CREATE TABLE PEDIDO(
  NUMPEDIDO INTEGER NOT NULL,
  EMISSAO DATE,
  CLIENTE VARCHAR(50));

ALTER TABLE PEDIDO ADD CONSTRAINT PK_PEDIDO PRIMARY KEY
(NUMPEDIDO);

CREATE TABLE ITEM (
  NUMPEDIDO INTEGER NOT NULL,
  ITEM INTEGER NOT NULL,
  PRODUTO VARCHAR(50),
  PRECO DOUBLE PRECISION,
  QUANTIDADE INTEGER);

ALTER TABLE ITEM ADD CONSTRAINT PK_ITEM PRIMARY KEY (NUMPEDIDO,
ITEM);
ALTER TABLE ITEM ADD CONSTRAINT FK_ITEM_PEDIDO FOREIGN KEY
(NUMPEDIDO)
  REFERENCES PEDIDO(NUMPEDIDO) ON UPDATE CASCADE ON DELETE
CASCADE;

COMMIT;

INSERT INTO PEDIDO(NUMPEDIDO, EMISSAO, CLIENTE)
    VALUES(1, '02.02.2009', 'JOAO DA SILVA');
INSERT INTO ITEM(NUMPEDIDO, ITEM, PRODUTO, PRECO, QUANTIDADE)
    VALUES(1, 1, 'PRODUTO A', 1.5, 2);
INSERT INTO ITEM(NUMPEDIDO, ITEM, PRODUTO, PRECO, QUANTIDADE)
    VALUES(1, 2, 'PRODUTO B', 0.5, 3);
INSERT INTO ITEM(NUMPEDIDO, ITEM, PRODUTO, PRECO, QUANTIDADE)
    VALUES(1, 3, 'PRODUTO C', 2.75, 1);
INSERT INTO ITEM(NUMPEDIDO, ITEM, PRODUTO, PRECO, QUANTIDADE)
    VALUES(1, 4, 'PRODUTO D', 10, 1);
INSERT INTO PEDIDO(NUMPEDIDO, EMISSAO, CLIENTE)
    VALUES(2, '03.03.2009', 'PEDRO SOUZA');
INSERT INTO ITEM(NUMPEDIDO, ITEM, PRODUTO, PRECO, QUANTIDADE)
    VALUES(2, 1, 'PRODUTO D', 10, 2);
INSERT INTO ITEM(NUMPEDIDO, ITEM, PRODUTO, PRECO, QUANTIDADE)
    VALUES(2, 2, 'PRODUTO C', 2.75, 5);
INSERT INTO PEDIDO(NUMPEDIDO, EMISSAO, CLIENTE)
    VALUES(3, '26.04.2009', 'MARIA JOAQUINA');
INSERT INTO ITEM(NUMPEDIDO, ITEM, PRODUTO, PRECO, QUANTIDADE)
    VALUES(3, 1, 'PRODUTO A', 1.5, 4);
INSERT INTO ITEM(NUMPEDIDO, ITEM, PRODUTO, PRECO, QUANTIDADE)
    VALUES(3, 2, 'PRODUTO B', 0.5, 10);
INSERT INTO PEDIDO(NUMPEDIDO, EMISSAO, CLIENTE)
    VALUES(4, '06.05.2009', 'LUIZ OLIVEIRA');
INSERT INTO ITEM(NUMPEDIDO, ITEM, PRODUTO, PRECO, QUANTIDADE)
    VALUES(4, 1, 'PRODUTO A', 1.5, 2);
INSERT INTO ITEM(NUMPEDIDO, ITEM, PRODUTO, PRECO, QUANTIDADE)
    VALUES(4, 2, 'PRODUTO B', 0.5, 3);
INSERT INTO ITEM(NUMPEDIDO, ITEM, PRODUTO, PRECO, QUANTIDADE)
    VALUES(4, 3, 'PRODUTO E', 5, 3);
COMMIT;
```

Montando a conexão com o banco de dados

Agora que já temos o banco de dados criado, vamos iniciar o desenvolvimento da aplicação. Inicie o Delphi e crie um novo projeto usando o menu File>New>VCL Forms Application – Delphi for Win32 salvando-o Proj_ReportBuilder.dpr na pasta que criamos anteriormente. Salve a unit principal com o nome untRelatorio.pas. Neste exemplo utilizarei o componente dbExpress para conexão, juntamente com o driver da DevArt para conexão com o Postgres e pode ser encontrado no site http://www.devart.com/pgdac/.

Insira no form um TSQLConnection(sCon). Adicione uma nova conexão ao banco de dados como mostra a **Figura 2** clicando duas vezes no componente e em seguida no botão '+'. Preencha os dados necessários para acesso ao banco de dados e confirme.

Adicione agora um TSQLDataSet(qryPedido). Configure a propriedade SQLConnection apontando-o para o sCon e configure também a propriedade CommandText inserindo a instrução SQL da **Listagem 2**. Perceba que estamos selecionando os campos das tabelas Pedido e Item utilizando um Left Join para unirmos ambas tabelas. Essa instrução pode ter pequenas variações caso mude de banco de dados.

Adicione agora um TDataSetProvider(dspPedido) ligando-o ao qryPedido pela propriedade DataSet. Inclua também um TClientDataSet(cdsPedido) com sua propriedade ProviderName apontada para o dspPedido. Clique duas vezes no cdsPedido e com o botão direito inclua todos os campos usando a opção Add All Fields. Por fim, arraste um DataSource(dsrPedido) para o formulário e aponte-o para o cdsPedido.

Com o exemplo será bem simples, não faremos telas de inclusão, alteração e etc. Portanto, apenas insira um Button(btnImprimir) com o Caption igual a Imprimir Relatório em nosso form e vamos começar a construir relatório. Veja a **Figura 3** como deve ficar nosso formulário.

Figura 3. Form e seus componentes

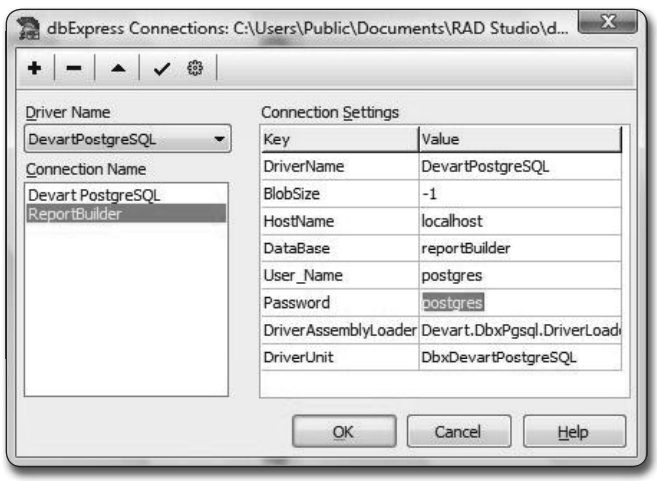


Figura 2. Criando conexão com o banco de dados

Listagem 2. Instrução SQL para seleção dos pedidos

```
SELECT
  PEDIDO.NUMPEDIDO,
  PEDIDO.EMISSAO,
  PEDIDO.CLIENTE,
  ITEM.ITEM,
  ITEM.PRODUTO,
  ITEM.QUANTIDADE,
  ITEM.PRECO,
  ITEM.QUANTIDADE * ITEM.PRECO AS TOTALITEM
FROM
  PEDIDO
  LEFT JOIN ITEM
  ON (PEDIDO.NUMPEDIDO = ITEM.NUMPEDIDO)
ORDER BY
  PEDIDO.NUMPEDIDO,
  ITEM.ITEM
```

Até aqui, preparamos o terreno para montar o nosso relatório. Agora na paleta do ReportBuilder, encontre e adicione os componentes TppDBPipeline(ppdbPedido) com a propriedade DataSource ligada ao dsrPedido.

Para termos acesso ao design do ReportBuilder e consequentemente desenharmos nosso relatório, precisamos de um componente TppReport(ppRelatorio). Na propriedade DataPipeLine do ppRelatorio, ligue-o ao ppdbPedido, este TppDBPipeline é quem comandará o relatório. Seu form ficará como na **Figura 3** como já mencionado.

Editor de Relatórios do ReportBuilder

Como dito no tópico anterior, quem nos dá acesso ao design de nossos relatórios no ReportBuilder é o componente ppRelatorio. Clicando duas vezes nele uma interface será aberta para que possamos desenhar literalmente o relatório como é mostrado na **Figura 4**. Como podemos ver, a janela do editor mostra na parte superior as barras de ferramentas com os componentes que poderão ser usados no relatório, tais como labels, memos, lines, etc, e também as barras de configuração de fontes e de alinhamentos. Caso alguma barra não esteja sendo mostrada, acesse o menu Visualizar > Barras de Ferramentas ou View>Toolbars se seu ReportBuilder estiver em inglês.

Antes de darmos continuidade ao exemplo, vamos entender um pouco sobre as barras de ferramentas, botões e funcionalidades do editor.

Na **Figura 5** podemos ver a barra de ferramentas Padrão. Nela temos os principais componentes para desenhar o relatório, tais como Label, Memo, Variaveis de Sistema, Image, etc. Vejamos uma rápida explicação de cada ícone da esquerda para a direita.

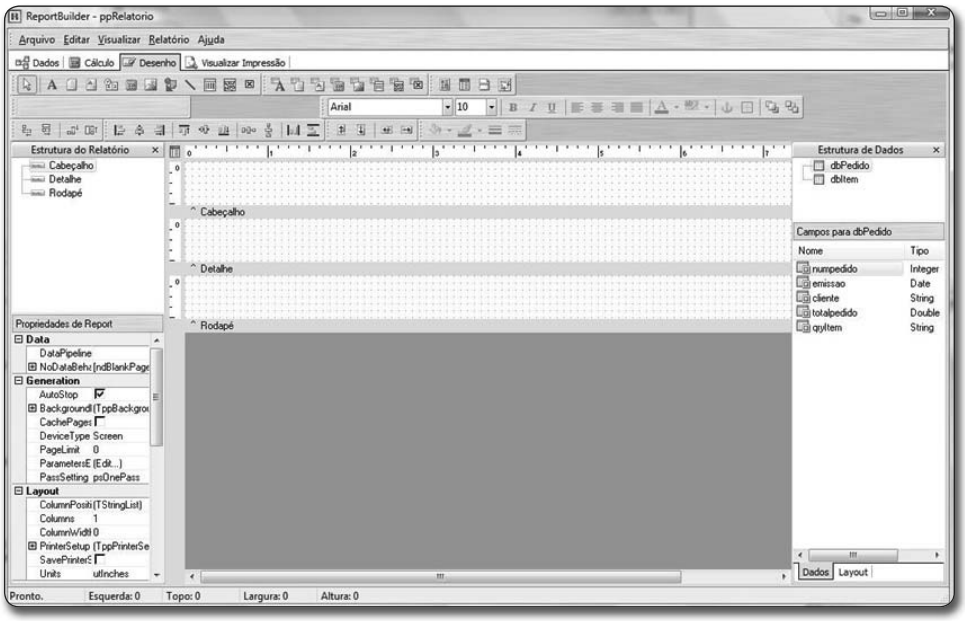


Figura 4. Editor de Relatórios do ReportBuilder

• Selecionar Objeto: utilizada para selecionar qualquer objeto dentro do editor;
• Texto: tem a função semelhante ao Label, ou seja, imprimir um texto;
• Memorando: funciona como o Memo, exibindo um texto com quebra de linha;
• Texto Formatado: assim como o Memorando, exibe um texto com quebra de linha, mas pode receber formatação;
• Variável do Sistema: a variável do sistema é um conjunto de valores que são alimentados ao longo do relatório tal como o número da página, o nome do documento, etc;
• Variável: funciona como uma variável no Delphi, você pode fazer cálculos que serão exibidos no relatório;
• Imagem: se necessitar colocar uma imagem no relatório, utilize o componente Imagem para carregar e posicionar a mesma;
• Polígono: desenho de forma como quadrados, retângulos e círculos;
• Linha: desenha um linha horizontal ou vertical;
• Código de Barras: imprime um código de barras. Existem diversos modelos como o Code39, Code128 e o EAN13;
• 2DBarCode: semelhante ao código de barras, só que utilizando a técnica de código de barras em 2 dimensões;
• CheckBox: imprime um CheckBox.

Na sequência vemos a barra de ferramentas de dados e a barra de controles avançados (**Figura 6**). Ela possui os principais controles que podem ser conectados ao banco de dados para exibição de informações provenientes de instruções SQL etc.

Figura 5. Barra de ferramentas padrão

Figura 6. Barra de ferramentas dados

Vejamos a funcionalidade de cada ícone:
• Campo de Texto: é um Texto ligado a um campo da tabela de nosso banco de dados;
• Campo Memorando: é um Memorando, listando valores a partir de um campo;
• Campo Texto Formatado: parecido com o Texto Formatado, mas com a função de semelhante ao documento modelo do Word;
• Campo Calculado: efetua somas, médias, contagens dos valores fornecidos pelo campo;
• Campo Código de Barras: imprime um código barra fornecido pelo campo;
• DB2DBarCode: imprimi um código de barras em 2 dimensões com valores fornecidos pelos campos do BD;
• DBCheckBox: imprime um CheckBox com ligação a um campo e estados para Checked e UnChecked.
• Região: funciona como um Panel para agrupar componentes;
• Sub-Relatório: exibe o novo relatório dentro do relatório atual, mas com uma fonte de dados diferente;
• Page Break: gera uma quebra de página;
• Referência cruzada: gera uma referência cruzada entre duas fontes de dados.

Existem, claro, outras barras como podemos ver no editor. É recomendável que se explore cada uma delas para que se tenha noção exata de suas funcionalidades. A esquerda do editor é mostrada a estrutura do relatório que consiste na listagem de cada banda e seus componentes (**Figura 7**). Logo abaixo a janela de propriedades dos objetos selecionados, semelhante ao

Object Inspector do Delphi (**Figura 8**). À direita fica a estrutura de dados que tem a função de mostrar todos os ppDBPipeLines que estão disponíveis e seus respectivos campos (**Figura 9**).

A área central do editor é reservada ao desenho dos relatórios. Para o desenhar o ReportBuilder utiliza-se de bandas onde cada uma tem a uma função específica:

• Banda de cabeçalho: tem a função de exibir um conjunto de objetos a cada nova página, sempre no topo da folha;

• Banda detalhe: que tem a função de exibir, cada registro que é fornecido pelo dbPipeLine, uma lista de objetos que em geral estão ligados ao banco de dados;

• Rodapé: é impresso na parte inferior de cada página, geralmente contêm o número da página e uma linha.

Abaixo de cada banda, existe uma faixa que é utilizada para redimensionar a banda, como mostro na **Figura 10**.

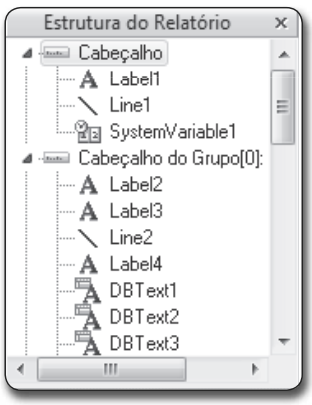


Figura 7. Estrutura de Dados

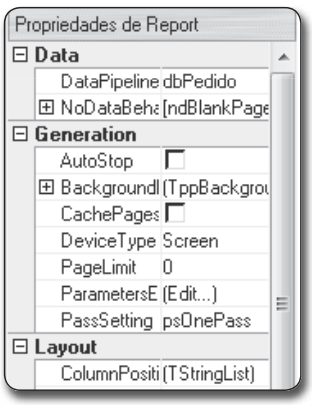


Figura 8. Propriedades do Objeto Selecionado

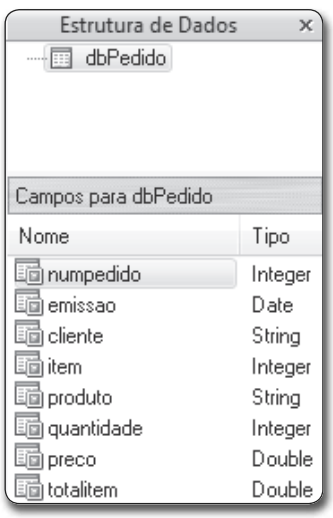


Figura 9. Estrutura de Dados

Nota

Note que no meu caso instalei a versão em Português do ReportBuilder, por isso toda a interface do meu editor e componentes estão com os nomes em nossa língua. Então fique atento, pois os nomes e a disposição de alguns controles mudam bastante na versão em inglês.

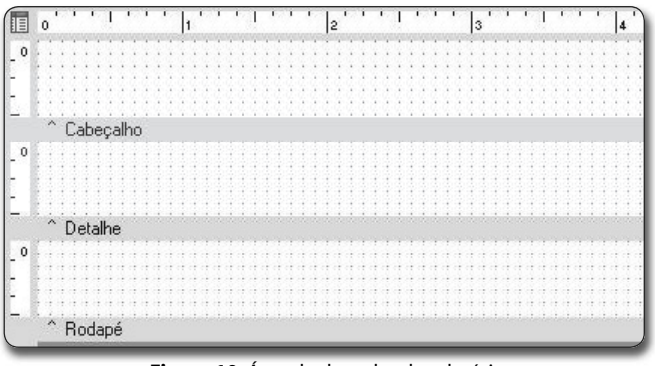


Figura 10. Área de desenho do relatório

Montando o relatório

Vamos começar o desenho relatório adicionando a banda de cabeçalho um componente de Texto(Label) na parte superior que será o nosso título. Altere a propriedade Caption para Relatório de Pedidos Emitidos. Adicione um componente de Variável do Sistema (SystemVariable) no canto superior direito e configure a propriedade VarType para vtDateTime. Adicione uma Linha(Line) abaixo do título. Diminua o tamanho da banda do cabeçalho.

Agora vamos adicionar um grupo ao relatório: entre no menu Relatórios>Grupos. Neste diálogo (**Figura 11**) iremos configurar no relatório uma quebra de grupo por Pedido, isso significa que cada vez que o número do pedido mudar será impressa a banda de Rodapé de grupo (exceto na primeira vez) e a banda de Cabeçalho de grupo.

No ComboBox do Grupo, selecione o campo NumPedido e clique no botão Adicionar. Será criada, automaticamente, duas bandas. Uma será o cabeçalho do grupo NumPedido e a outra o rodapé. Clique em Ok para fechar o diálogo. Na banda cabeçalho do grupo criado aumente-a um pouco e adicione três Textos com os seguintes Caption's: Pedido, Data de Emissão e Cliente, assim como mostra a **Figura 12**.

Adicione logo abaixo uma Linha e na sequência inclua três Campos de Textos(DBText), configurando a propriedade DataField para cada um deles com: NumPedido, Emissao e Cliente. O campo de texto é responsável por exibir os dados de cada campo do relatório. Logo abaixo, adicione mais uma linha. Ainda na banda do grupo, adicione cinco textos com os dizeres: Item, Produto, Quantidade, Preço e Total e mais uma vez arraste uma linha.

Pronto, nossa banda de cabeçalho está pronta. O que precisamos agora é configurar a banda detalhe. Como mencionado, a banda detalhe fará a impressão dos dados variáveis do relatório. Ela repetirá a impressão até que todos os registros da base de dados sejam impressos.

Agora na banda detalhe adicione cinco DBText configurando as propriedades DataField com os campos item, produto, quantidade, preco e totalItem. Clique com botão direito sobre os objetos do preco e totalItem e clique sobre a opção Formato de Apresentação. Nesta caixa de diálogo (**Figura 13**), selecione a máscara #,0.00;-#,0.00 e clique em Ok . Diminua a altura da banda detalhe.

Nota

A lista de máscaras é mostrada de acordo com o tipo do campo, experimente clicar sobre o campo de texto da emissão. Verá que outras máscaras ficam disponíveis.

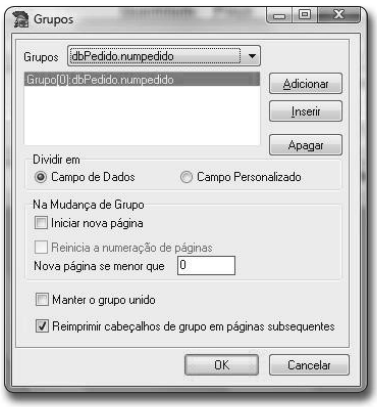


Figura 11. Diálogo de Criação de Grupos

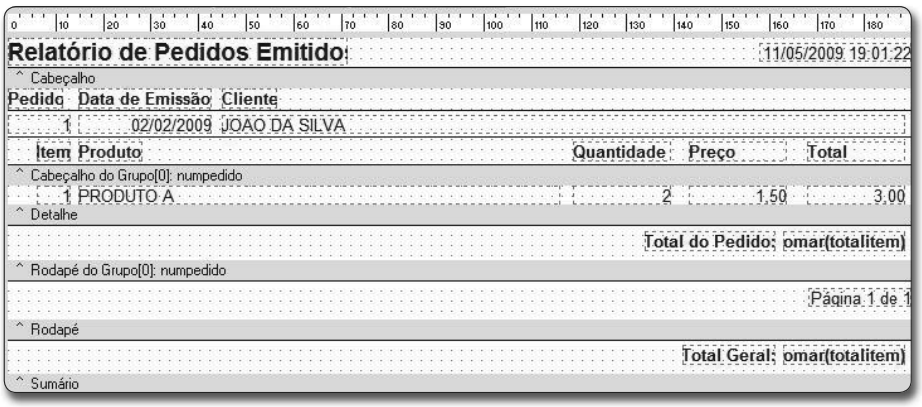


Figura 12. Relatório montado

Figura 13. Editor de Formatação de Campos

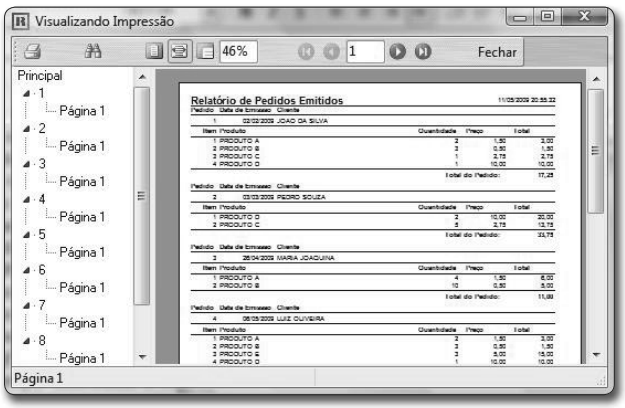


Figura 14. Pré-visualização do Relatório

Vamos agora configurar o rodapé do grupo, adicionando um Texto com o Caption Total do Pedido:. Vamos arrastar também um Campo Calculado (DBCalc). O Campo Calculado efetua somatórias, contagens, médias dos valores nele configurados. O padrão do controle é soma. Configure a propriedade DataField para totalItem. Neste caso ele irá somar, para cada registro do relatório, o valor do campo totalitem. Quando o DBCalc é colocado em uma banda de rodapé de grupo, ele irá somar todos os itens do grupo. Assim que o grupo mudar, será reiniciada a soma.

Clique com botão direito e clique sobre a opção Formato de Apresentação e selecione uma máscara para o campo. Adicione uma linha acima do total. Na banda rodapé, insira uma Variável do Sistema e configure a propriedade VarType para vtPageSetDesc. Novamente inclua uma linha para fazer a separação.

Faremos agora a configuração da banda Sumário. Esse tipo de banda é usado para exibir resumos de nossos relatórios. Ela é impressa somente na última folha do relatório antes do rodapé.

Para isso vamos no menu Relatório>Resumo. Nesta banda vamos colocar os mesmo componentes que foram incluídos na banda rodapé do grupo. O componente Campo Calculado que será colocado, por estar na banda de Sumário, irá somar todos os valores do campo configurados até que o relatório chegue ao final.

Agora feche o editor do ReportBuilder e clique duas vezes sobre o botão btnImprimir e adicione o código da **Listagem 3**. Veja como é simples fazer a chamada ao relatório.

Apenas chamamos o método Print do componente ppRelatorio. As demais linhas são utilizadas para abrir o ClientDataSet que fará a captura dos dados. Pressione a tecla F9 para rodar a aplicação e então clique no botão para visualizar o relatório (**Figura 14**).

Listagem 3. Impressão do relatório

```
procedure TForm1.btnImprimirClick(Sender: TObject);
begin
  if cdsPedido.Active then
    cdsPedido.Close;
  cdsPedido.Open;
  ppRelatorio.Print;
end;
```

Conclusão

Como podemos ver ao longo do artigo, a utilização do ReportBuilder é bastante simples e não requer nenhuma configuração especial para construir relatórios. Esse foi um exemplo simples, mas é de extrema importância o estudo contínuo da ferramenta, já que existem inúmeras funcionalidades que não vimos aqui.

Vimos como montar um relatório simples, mas em breve falarei mais sobre alguns recursos avançados como sub-relatórios e gráficos. Até a próxima. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos para iniciantes na linguagem Delphi

Introdução à POO

Aprenda, de uma forma simples, a programar da forma correta

Resumo do DevMan

Este é o primeiro artigo de uma série sobre POO com Delphi. Vamos, em um primeiro momento, tratar da orientação a objetos na teoria, fazendo analogias e comparações com o mundo real. Veremos quais são as bases da OO, no que é fundamentada. Na sequência, vamos iniciar um exemplo prático, onde você aprenderá a criar classes, com atributos e métodos, e como instanciar objetos.

Nesse artigo veremos

Este artigo aborda a orientação a objetos com o Delphi, usando uma metodologia simples, didática, de fácil aprendizado. Veremos na teoria, e também na prática, todos os conceitos, fundamentos e recursos oferecidos pelo Delphi para promover a POO.

Qual a finalidade

A POO pode e deve ser aplicada de forma inteligente, ela serve para construir sistemas mais robustos, mais confiáveis, de fácil manutenção, que permitam maior reaproveitamento de código.

Quais situações utilizam esse recurso?

A POO é útil em qualquer sistema, seja ele Web, Desktop, Mobile, não importa o tipo de aplicação. Os conceitos aqui apresentados, em um exemplo simples, podem ser utilizados em aplicações reais, como é apresentado em um estudo de caso no final desta série.

Guinther Pauli
guinther@devmedia.com.br
Tem 21 anos de experiência na área de desenvolvimento e arquitetura de software, dos quais 14 dedicados ao Delphi e 9 ao .NET. Certificado Borland em Delphi 6, 7, 2005, 2006, Delphi para Web e Kylix, Certificado Microsoft MCP, MCAD, MCSD. NET, MCTS e MCPD. Editor geral da Revista .NET Magazine Brasil e professor de Pós-Graduação. Publicou mais de 150 artigos, 400 vídeo-aulas, palestrante em 5 edições da Borland Conference. Especialista em desenvolvimento Delphi, C#, .NET, ASP.NET, atuando também em projetos de migração orientando empresas e desenvolvedores na adotação de melhores práticas.

O objetivo deste artigo é apresentar, de uma forma simples e prática, os principais conceitos da programação orientada a objetos com o Delphi. É um artigo voltado ao público iniciante, tanto que utilizarei uma abordagem bastante didática e fácil, ao invés de aplicar os conceitos em um exemplo complexo. Aprendendo o básico sobre os principais recursos da POO, entendendo como o Delphi permite a orientação a objetos em uma linguagem elegante, permitirá a você aplicar futuramente os seus conhecimentos em projetos reais.

Você aprenderá como criar classes, instanciar objetos, usar a herança, abstração e polimorfismo. Definirá propriedades, entenderá os diferentes especificadores de visibilidade (private,

public, protected e published), criará métodos de classe, get/set, estáticos e abstratos, construtores e destrutores. Conhecerá importantes palavras-chave da Delphi Language usadas na POO. Aprenderá a dar um passo além da POO, ao transformar classes em componentes, a utilizar pacotes para organizar e agrupar units, verificar dependências e muito mais.

Princípios básicos da Orientação a Objetos

A Orientação a Objetos está fundamentada em quatro pilares: abstração, encapsulamento, herança e polimorfismo. Mesmo para aqueles que desenvolvem há anos com o Delphi, esses conceitos podem passar até despercebidos. Ao contrário de linguagens totalmente orientadas a objeto, como Java e C#, o Delphi não obriga que você desenvolva um sistema usando técnicas de OO.

O apelo RAD da ferramenta faz com que os desenvolvedores construam seu software parcialmente ou totalmente orientado a eventos (EDP - Event-Driven Programming). É muito simples colocar um ClientDataSet no DataModule, criar um manipulador para o evento OnNewRecord e inserir código para inicializar dados. Da mesma forma, é simples inicializar controles de um formulário no seu evento OnShow. É possível que você tenha criado um sistema completo sem sequer ter criado uma única classe! Veja que o Delphi cria uma classe para o formulário, uma para o DataModule etc. Mas e você, cria uma classe para processar regras de negócio?

É impossível falar em orientação a objetos sem recorrer ao mundo real, e sem recorrer à teoria (não se preocupe com a parte teórica, vamos ver tudo isso na prática mais adiante), o que consequentemente também nos obrigará a fazer muitas analogias. De fato, muito da OO é uma cópia fiel do que temos ao nosso redor. Um carro é um objeto, composto de outros objetos, que podem ser trocados. Um carro possui atributos, como sua cor, tamanho, modelo, número de portas, ano de fabricação, combustível, autonomia etc. Então, nada melhor que usar a vida real para entender a OO, técnica que usarei exaustivamente durante todo esse artigo.

Antes de começar, vamos rever aquilo que aprendemos no primeiro dia de aula de programação, a diferença entre classe e objeto. A analogia perfeita, uma classe é uma "receita de bolo". Objetos são os "bolinhos" que você faz a partir desta receita. TForm é uma classe. Form1, ou FormCliente, é um objeto, uma instância real da classe. Podemos ter várias instâncias de uma mesma classe. No Delphi, por exemplo, temos em uma aplicação com vários ClientDataSets, mas todos construídos a partir da classe TClientDataSet.

Vamos agora examinar a definição dos quatro pilares básicos de qualquer linguagem orientada a objetos:

Abstração: capacidade de representar conceitos do domínio do problema, ressaltando apenas o que for relevante para a aplicação em questão. A definição da classe em Delphi atende a esse requisito, pois ela é um modelo limitado de alguma entidade real, que implementa apenas as características que são necessárias para a aplicação, sendo desenvolvida em um dado contexto.

Encapsulamento: a linguagem deve permitir a definição de módulos auto-suficientes, possibilitando a implementação do conceito de ocultação de informação (information hiding), ou seja, esconder os detalhes internos. O uso dos especificadores de visibilidade (private e protected, por exemplo) e o comportamento dos objetos em Delphi reforçam esse princípio, principalmente se o desenvolvedor usar propriedades nas classes, como forma de acesso aos campos internos.

Herança: mecanismo que permite a definição de uma hierarquia de classes, com o objetivo de reutilizar características já definidas e estendê-las em níveis maiores de detalhes ou especialização. O Delphi implementa um mecanismo eficiente de herança simples (apenas uma classe ancestral – não há suporte para herança múltipla de classes).

Polimorfismo: classes descendentes podem implementar de forma diferente um conjunto de operações comuns definidas na classe base (no Delphi, usamos as diretivas virtual e override para aplicar o polimorfismo).

Existem também algumas "regras" quando desenvolvemos de forma orientada a objetos. São boas práticas que devem ser seguidas, para construir um software melhor, com boa qualidade, de fácil manutenção, com código que possa ser facilmente reutilizado em outros sistemas.

Quando falamos em encapsulamento, dizemos que uma classe deve ser o mais independente possível de outras classes. Imagine, por exemplo, que se toda vez que sua TV apresentasse um defeito, você tivesse que levar o aparelho de DVD junto para o conserto? Na POO, isso seria um problema grave de modelagem. Classes devem depender o mínimo de outras classes, para permitir uma fácil reutilização. Por esse motivo, elas também devem desempenhar uma única função.

Outra analogia. Em qualquer framework de acesso a dados bem modelado (como o ADO.NET), temos uma classe responsável pela execução de comandos SQL no banco de dados (nesse caso, o SqlCommand). E só! A conexão com o banco de dados, que é outra função, é desempenhada por outra classe (nesse caso, o SQLConnection). Para transações, outra classe. Para armazenar os dados lidos, outra classe. E assim por diante. Cada classe tem sua função, única, reaproveitável.

Mas classes não trabalham sozinhas, elas precisam ser ligadas, geralmente isso acontece por associação. Um objeto que executa comandos SQL no banco é inútil sem um objeto de conexão. Essa necessidade nos leva a outra regra: classes devem ser ligadas por um único ponto. No exemplo do ADO.NET, o objeto SqlCommand possui uma propriedade, um único ponto de ligação, que aponta para um objeto de conexão SqlConnection. E geralmente, esse ponto de ligação usa um tipo mais genérico, uma classe base, uma interface, que permita que outros tipos de objeto com a mesma função, mas diferente implementação, possam ser conectados.

O que isso significa? Novamente, vamos ao mundo real. Um computador é fabricado hoje. Ok. Porém, como criar entradas de conexão para ligar diferentes tipos de periféricos, alguns inclusive que ainda nem foram inventados? É definido um padrão, um protocolo, uma regra. O computador tem uma

ou mais entradas USB, e qualquer tipo de objeto que venha a ser criado, que queira se conectar a um micro, seguirá esse padrão. Assim surgem impressoras USB, Pen-Drives, Mouses etc. O ponto de ligação é único. Se estragou, você retira o objeto e coloca outro. Assim é a vida real, assim é a POO. Imagine se existisse um tipo de conexão para cada tipo de periférico? Provavelmente seu Notebook teria mais furos nas laterais que teclas no teclado. E quando inventassem um novo dispositivo, teria que levar na fábrica para instalarem uma nova porta de conexão. Não é assim que funciona na vida real e não é assim na POO.

Vagões de trem, outro exemplo clássico. Eles são conectados por um único ponto. Podem ser desconectados facilmente, outros modelos de vagões podem ser conectados, desde que o ponto de conexão siga um padrão. Se você já desenvolveu relatórios no QuickReport, reparou que ele não aponta para um tipo específico de objeto de dados. Ele aponta para um tipo genérico, no caso, TDataSet. Por quê? É o mesmo princípio da USB. Dessa forma, um relatório pode trabalhar com diferentes mecanismos de acesso a banco de dados, mesmo aqueles que ainda possam ser inventados. A única regra é que, para que um objeto de manipulação de dados possa servir de fonte de dados para um relatório, ele deve seguir a regra, mais especificamente, ser um descendente de TDataSet e implementar sua funcionalidade.

Quando falamos em herança, estamos falando obrigatoriamente em reutilização. Criamos uma classe básica, que possui as características comuns a todos os seus descendentes. Depois, vamos especializando essa classe, criando descendentes, adicionando características específicas. Por exemplo, na vida real, temos os meios de transporte. Isso poderia ser uma classe, se modelado no mundo OO, seria TMeioTransporte. Nessa classe, colocaríamos atributos e funcionalidades a todos os meios de transporte. Poderíamos definir a capacidade de se locomover, andar, movimentar. Na OO, isso daria origem a um método, Movimentar. Isso é comum a todos os meios de transporte. Depois, poderíamos criar classes mais específicas, como TCarro, TBarco, TBicicleta, TMoto, TTrem, TCarroca, TAviao.

Mas veja que cada meio de transporte se movimenta de uma forma diferente. Um carro se movimenta de forma bem diferente de um avião, claro. Fazer a mesma coisa, mas de formas diferentes, traz alguma coisa a sua mente? Se pensou polimorfismo, acertou em cheio.

Nota do DevMan

No Delphi, iniciar o nome de classes com um "T" é um padrão. T vem de type, ou tipo. O mesmo acontece para interfaces, todas começam com "I", por exemplo, IUnknown, IInterface etc. Aliás, se quiser ver como interfaces podem ser usadas ao extremo na POO, dê uma olhada no código fonte do WebSnap, no diretório de mesmo nome dentro dos fontes da VCL do Delphi. No .NET, as classes não seguem esse padrão, de ter o "T" para definir um tipo classe (ex.: DataSet, SqlConnection, StringBuilder, XmlReader), o que não vale para interfaces, que levam "I".

Muito bem, estamos começando a entender a POO. Mas ela é muito bonita na teoria, mas na prática, funciona? Por que implementar uma classe chamada DAL (Data Access Layer), criar nela métodos para manipular informações no banco de dados, instanciar objetos de acesso a dados, definir classes para mapear tabelas do banco, se eu posso simplesmente largar um ClientDataSet e... Deixa pra lá. O que proponho é mostrar a POO na prática, mas de uma forma extremamente SIMPLES. Eu não vou propor aqui a criação de um grande sistema financeiro, ou hospitalar, ou acadêmico. Ao invés disso, vamos usar objetos simples, como carros e pessoas. Você vai ver como é divertido usar a POO para resolver problemas cotidianos, e melhor do que isso, ao aprender a verdadeira POO no exemplo que faremos, saberá como resolver os seus próprios problemas e dos seus clientes em seus sistemas reais. Vamos lá!

Exemplo prático (e simples)

No Delphi, inicie uma nova aplicação VCL Win32. Dê o nome de FrmExemplo ao formulário e salve-o nomenado a unit como uFrmMain.pas. Dê o nome de ClassesObjetos ao projeto. Crie o formulário mostrado na **Figura 1** (o texto dentro dos Edits indica o nome que você deve dar a eles). Esse formulário servirá para entrarmos com os dados que serão usados para popular objetos que criaremos a seguir. Observe que ele possui duas sessões, delimitadas por dois GroupBoxes. Os Edits serão usados para o usuário informar os valores para os atributos dos objetos. Os botões vão criar e liberar os objetos da memória.

Figura 1. Formulário exibe as propriedades dos objetos

Vamos agora criar algumas classes, mas especificamente, uma classe para representar um carro e uma para representar um avião. Clique em File>New>Unit. Salve a nova unit com o nome de uCarro.pas. Usando o mesmo procedimento crie uma unit chamada uAviao.pas. Veja na **Listagem 1** o código das units. Observe a forma como declaramos uma classe no Delphi. Veja também que cada classe define atributos. Por exemplo, um carro tem uma capacidade.

Listagem 1. Código das units

```
unit uCarro;

interface

type
  TCarro = class
    Descricao : string;
    Capacidade : integer;
    Quilometragem : integer;
  end;

implementation

end.
-------------------------------------------------
unit uAviao;

interface

type
  TAviao = class
    Descricao : string;
    Capacidade : integer;
    HorasVoo : integer;
  end;

implementation

end.
```

Boa Prática

Seguindo um padrão da maioria das linguagens orientadas a objeto, procure sempre declarar uma classe por unit. Mas veja, no entanto, que a própria VCL do Delphi não segue esse padrão.

Volte ao formulário principal e adicione as units uCarro e uAviao à cláusula uses da interface, para podermos ter acesso às classes definidas. Agora declare duas variáveis na seção public do formulário:

```
public
  Carro : TCarro;
  Aviao : TAviao;
```

Nota

Nota: public é um especificador de visibilidade, conceito que discutiremos mais adiante. Por enquanto, basta entender que o que está declarado dentro da sessão public pode ser acessado sem restrição em qualquer parte do código.

Essas variáveis representarão instâncias das classes que acabamos de criar, e serão manipuladas pelos controles de tela do formulário principal. No botão Criar (do objeto Carro) digite o código da **Listagem 2**. Como o próprio nome sugere, o botão criará uma instância do tipo TCarro, atribuindo isso ao objeto chamado Carro. Isso é feito chamando o construtor da classe, um método especial chamado Create. Ainda no código, após criar o objeto, inicializamos seus atributos com base nos valores digitados na tela.

Listagem 2. Código do botão Criar Carro

```
procedure TFrmExemplo.BtnCriarCarroClick(Sender: TObject);
begin
  // cria o objeto e inicializa campos conforme valores dos
edits
  Carro:=TCarro.Create;
  if EdtDescCarro.Text<>'' then
     Carro.Descricao:=EdtDescCarro.Text;
  if EdtCapCarro.Text<>'' then
     Carro.Capacidade:=StrToIntDef(EdtCapCarro.Text,0);
  if EdtQuilometragem.Text<>'' then
     Carro.Quilometragem:=StrToIntDef(EdtQuilometragem.Text,0);
end;
```

Agora vamos fazer o mesmo para objetos do tipo TAviao. No botão Criar (do objeto Avião) digite o código da **Listagem 3**. O que fazemos aqui é a mesma coisa que fizemos para o carro, instanciamos uma classe e inicializamos seus atributos conforme os dados que o usuário digitar na tela.

Listagem 3. Código do botão Criar Avião

```
procedure TFrmExemplo.BtnCriarAviaoClick(Sender: TObject);
begin
  // cria o objeto e inicia campos conforme valores dos edits
  Aviao:=TAviao.Create;
  if EdtDescAviao.Text<>'' then
     Aviao.Descricao:=EdtDescAviao.Text;
  if EdtCapAviao.Text<>'' then
     Aviao.Capacidade:=StrToIntDef(EdtCapAviao.Text,0);
  if EdtHorasVoo.Text<>'' then
     Aviao.HorasVoo:=StrToIntDef(EdtHorasVoo.Text,0);
end;
```

Quando chamamos o Create, o objeto passa a ocupar um espaço na memória. Isso porque um objeto é uma instância real de uma classe. Após ser utilizado, ele precisa ser destruído. Já cuidamos da criação dos objetos, usando Create. Agora vamos cuidar da destruição. Isso pode ser feito chamando o método Free do objeto. Então no botão Liberar (do objeto Carro) digite o código da **Listagem 4**. Da mesma forma, para o avião, no botão Liberar digite o código da **Listagem 5**.

Listagem 4. Código do botão Liberar Carro

```
procedure TFrmExemplo.BtnLiberarCarroClick(Sender: TObject);
begin
  Carro.Free; // ou FreeAndNil(Carro)
end;
```

Listagem 5. Código do botão Liberar Avião

```
procedure TFrmExemplo.BtnLiberarAviaoClick(Sender: TObject);
begin
  Aviao.Free; // ou FreeAndNil(Aviao)
end;
```

Pronto! Já podemos testar as funcionalidades básicas implementadas até aqui. Rode a aplicação com F9 e faça os testes. Informe os valores para os atributos do Carro e clique em Criar. Depois libere-o. Estamos dando nossos primeiros passos na POO.

Criando métodos

Como você pode ter visto, até agora criamos apenas as classes com atributos, ou seja, definimos as características de um carro, de um avião. Não tratamos ainda de outro importante papel de qualquer classe, a funcionalidade, seu comportamento. Enquanto atributos e propriedades definem as características de uma classe, métodos servem para definir o que uma classe pode fazer. Por exemplo, um carro pode entrar em movimento. Vamos então criar um método chamado Mover. Abra a declaração de TCarro e declare o método Mover, como mostrado na **Listagem 6**.

Nota do DevMan

Ponteiro é um importante conceito de qualquer linguagem de programação, seja ela orientada a objeto ou não. Uma variável, um objeto, é uma estrutura que ocupa espaço na memória. Quando declaramos uma variável do tipo integer, por exemplo, um espaço é alocado na memória imediatamente, dependendo do escopo desta variável (se é de um procedimento, de uma unit, de uma classe). Porém, estruturas mais complexas, como classes, que possuem vários atributos e ocupam mais espaço na memória, não são alocadas automaticamente na memória. Precisamos criar instâncias dessas classes, criar os objetos, em tempo de execução. Ou seja, a alocação de memória é dinâmica.

Esse conceito surgiu há muito tempo. A ideia é simples. Declaramos uma estrutura, que guardará informações sobre um cliente, mas não alocamos o espaço em memória para alimentar essa estrutura até que seja realmente necessário. Ao invés disso, criamos uma variável que aponta (daí o nome ponteiro) para outra variável que representa essa estrutura. No Pascal antigo, usávamos o símbolo ^ para representar um ponteiro. Então, um ponteiro é basicamente um número, um endereço de memória, que você pode usar para referenciar uma outra variável, que será realmente criada em outro momento. O verbo referenciar usado na frase anterior não foi por acaso. Variáveis que apontam para outras variáveis são conhecidas na programação como variáveis de referência.

No Delphi, ainda existe o conceito de ponteiros, mas não precisamos trabalhar diretamente com eles. Por exemplo, Form1 é um "ponteiro". A estrutura da classe TForm1 só vai ser usada quando o método Create for chamado. Antigamente, métodos como GetMem, AllocMem etc. eram usados para alocar memória reservada por ponteiros. Quando um objeto, que na verdade é um ponteiro, não está apontando para nada, ele possui o valor NIL (nulo, ou null, como usado em outras linguagens).

FreeAndNil, recurso criado no Delphi 5 e citado no comentário do código das Listagens 4 e 5, pode ser usado para automaticamente liberar um objeto e fazer com que o ponteiro seja anulado, não apontado para uma posição de memória onde não existe mais o objeto real.

Listagem 6. Declarando o método Mover em TCarro

```
TCarro = class
  Descricao : string;
  Capacidade : integer;
  Quilometragem : integer;
  procedure Mover;
end;
```

Aperte Shift+Ctrl+C para que o Delphi gere o cabeçalho da implementação para o método que acabamos de definir. Implemente o método como mostrado na **Listagem 7**.

Listagem 7. Implementação do método Mover de TCarro

```
procedure TCarro.Mover;
begin
  ShowMessage(Descricao+' entrou em movimento.');
end;
```

Declare Dialogs na cláusula uses da unit uCarro, para que possamos usar o ShowMessage.

Agora vamos permitir que o usuário, após criar um objeto Carro usando o formulário, possa colocá-lo em movimento, chamando seu método Mover. No formulário principal, nas opções do objeto carro, coloque então um botão com o Caption Mover. No seu evento OnClick digite o código da **Listagem 8**.

Listagem 8. Chamando o método Mover do Carro

```
procedure TFrmExemplo.BtnMoverCarroClick(Sender: TObject);
begin
  Carro.Mover;
end;
```

Agora repita os mesmos passos para classe TAviao. Na **Listagem 9** veja a implementação do método Mover para TAviao.

Listagem 9. Implementação do método Mover de TAviao

```
procedure TAviao.Mover;
begin
  ShowMessage(Descricao+' está voando.');
end;
```

Execute a aplicação. Preencha os Edits com dados para criar os objetos, crie-os e a seguir coloque-os em movimento clicando nos respectivos botões. A **Figura 2** mostra o resultado obtido até aqui.

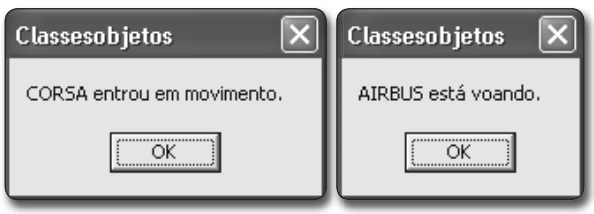


Figura 2. Chamando um método

Conclusão

Muito bem, isso finaliza a primeira parte desta série. Ainda é pouco sobre tudo o que ainda vamos ver sobre OO com Delphi. Mas o que foi visto aqui é fundamental: aprendemos a declarar classes, com atributos e métodos. Vimos como instanciar essas classes, que a partir desse processo, geram objetos. Nos objetos, preenchemos seus atributos com valores para definir suas características e chamamos métodos para testar sua funcionalidade.

Nos próximos artigos, vamos aprofundar, e muito, os conceitos da POO. Você já pode notar, por exemplo, que ambas as classes que criamos – TCarro e TAviao – possuem características em comum, como a descrição e capacidade. Além disso, ambas as classes tem comportamentos semelhantes, como a capacidade de se movimentar. Isso nos levará a dois importantes conceitos da orientação a objeto, a herança e polimorfismo, assuntos que discutiremos na próxima edição.

Um grande abraço e até lá. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos sobre a linguagem PHP e a ferramenta Delphi for PHP

POO, PHP, DAO e MySQLI

Conheça os princípios básicos da POO e veja um exemplo real de aplicação

Resumo do DevMan

POO – Programação Orietada a Objetos certamente é o tema que mais chama a atenção de todos. Por que será? A explicação é simples: com POO o sistema torna-se mais robusto, inteligente, de rápido manutenção e sem dúvida nenhuma melhor entendível. Podemos aplicar POO em todas as linguagens de programação.

Veremos como trabalhar com POO em PHP puro, criando classes, atributos, métodos e utilizando um dos conceitos mais interessantes o DAO – Data Access Objects para separarmos regras de negócios de camadas de acesso a dados e layout.

Nesse artigo veremos

Veremos o desenvolvimento de uma pequena aplicação para que possamos entender como funcionam Classes, Atributos, Métodos, Herança e Instância;

Qual a finalidade

Desenvolver uma aplicação com o intuito de praticarmos o desenvolvimento de sistemas com POO no PHP;

Quais situações utilizam esse recurso?

Como já mencionado em diversos os artigos e como é sabido da maioria dos desenvolvedores, criar sistemas seguindo os princípios da orientação a objetos é uma das melhores práticas que podemos empregar em qualquer sistema.

Rafael Silva
rafael@rlsystem.com.br
Trabalha com análise e desenvolvimento de sistemas. Atua na área de desenvolvimento desktop utilizando Delphi, C#, e foco na web com PHP, ASP.NET, CSS, AJAX, sempre utilizando os padrões da Web 2.0 em seus projetos. Utiliza banco de dados PostGreSQL, MySQL, SQL Server e Firebird.

Na edição passada, vimos o principais conceitos da utilização e deenvolvimento de sistemas com POO no PHP. Vimos como criar classes, herança, o que são atributos, métodos e uma série de outras informações pertinentes a um projeto POO. De fato o tema POO parece nunca cair de moda e a razão é bem simples. Com orientação a objetos poupamos tempo, melhoramos o entendimento dos sistemas, reutilizamos código evitando a perda de tempo (já mencionada). Uma montanha de recursos e vantagens podem se mencionadas aqui e em outros artigos. Realmente o tema é extremamente longo e detalhado.

A idéia desse artigo é colocar em prática tudo que aprendemos no artigo anterior. Faremos a criação de um sistema que nos possibilitará entender perfeitamente tudo o que já foi dito. Mão na massa!

Exemplo real de uso

Sim, foram mostrados vários conceitos, exemplos, e tenho certeza que sempre fica uma dúvida, isso é normal. Temos que nos aprofundar um pouco mais nisso, para isso vamos criar uma aplicação(Contas Pagar e Receber) em OO, utilizando logicamente PHP, banco de dados MySQL, e o mais interessante a extensão MySQLI. A principal idéia aqui é também utilizar a DAO (Data Access Objects), para a persistência de dados, separando as regras de negócios das regras de acesso a banco de dados.

Lembrando que aqui não estaremos utilizando nenhum editor. Você pode escolher um de sua preferência. Apenas crie um diretório a sua escolha para que possa ser salvo todos os arquivos que criaremos nesse projeto. É interessante também você ter um servidor instalado na sua máquina, rodando PHP e MySQL e caso não tenha nenhum recomendo a utilização do WAMP (um pacote que faz a instalação do apache, php, mysql em alguns cliques).

Nota do DevMan

A extensão MySQLI, conhecida como MySQL melhorada, foi desenvolvida para dar suporte as novas funcionalidades do MySQL 4.1. Outra coisa interessante é que com ela podemos utilizar ela no modo orientado a objetos, já que não verdade ela é uma classe.

Banco de Dados

Primeiramente vamos criar nosso banco de dados. Não entrarei em detalhes aqui sobre isso por não ser o escopo principal do artigo. Basicamente teremos quatro tabelas: Clientes, Fornecedores, ContasPagar e ContasReceber. Criaremos também um sistema simples, mas com base nele será possível sim e, com facilidade, você incrementar mais funcionalidades ao sistema. É importante salientar também algumas regras da nossa aplicação, onde uma conta a pagar, será paga a um fornecedor e conta a receber será recebida de um cliente, com essa pequena idéia você já vai entender as tabelas do banco de dados. Conforme a **Figura 1**, crie as tabelas necessárias.

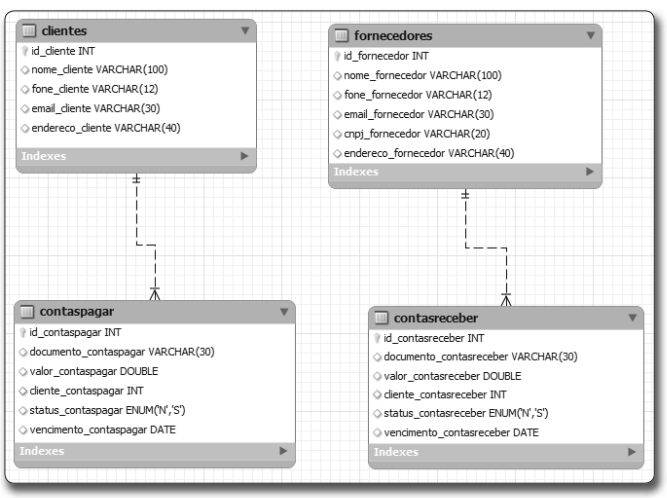


Figura 1. Tabelas do banco de dados

ClubeDelphi PLUS!

Acesse agora o mesmo o portal do assinante ClubeDelphi e inscreva-se para o curso Rodrigo Carreiro que mostra como trabalhar com bancos de dados MySQL.
www.devmedia.com.br/cursos/listcurso.asp?curso=136

Classe Contas Pagar e Contas Receber

Para iniciar nosso desenvolvimento, termos que começar criando as classes ContasPagar e ContasReceber. Essas classes serão responsáveis por abstrair as tabelas do nosso banco de dados, que irão conter nossos atributos e propriedades com métodos GET e SET, que são os campos das suas respectivas tabelas.

Inicie um novo arquivo no Bloco de Notas ou no seu editor preferido. Digite o código da **Listagem 1**. Crie no diretório principal da aplicação um subdiretório denominado Classes e salve as respectivas classes no mesmo com os nomes de ContasPagar.php e ContasReceber.php.

Como dito, na **Listagem 1** temos o código das classes Contas a Receber e Contas a Pagar. No início da listagem estamos definindos os atributos da classe Conta a Receber. Veja que criamos 6 atributos que são:

- $id_contasreceber: id do registro;
- $documento_contasreceber: documento do contas a receber;
- $valor_contasreceber: evidentimente o valor desse documento;
- $cliente_contasreceber: id do cliente para que possamos identificá-lo;
- $status_contasreceber: o status da documento, ou seja, o andamento do documento;
- $vencimento_contasreceber: vencimento do registro.

Perceba que todos os atributos estão declarados como Private, ou seja, estarão disponíveis apenas para classe. Em seguida temos a declaração de todos os métodos Get e Set das classes. Como mencionado, esse métodos servem para que possamos recuperar ou gravar um valor no sistema. Tanto a classe ContasReceber quanto ContasPagar possuem atributos e métodos semelhantes. O importante a observar é que a construção é idêntica para ambos os casos.

Conexão usando a extensão MySQLI

Antes de começarmos a criar nossas classes DAO, criaremos, no diretório principal do projeto, um arquivo denominado Conexao.php que será responsável pela conexão com nosso banco de dados. Para isso vamos criar uma variável que será uma instância da classe MySQLI e no seu próprio construtor já informar os parâmetros como (servidor, login, senha, banco) com isso podemos utilizar apenas o objeto $con que será do tipo MySQLI como citei para realizar a conexão com o BD. Nós visualizaremos melhor isso tudo com as classe DAO.

Nota

Para que possa usar a extensão MySQLI é necessário habilitá-la no arquivo PHP.INI que fica no diretório de instalação do PHP. Basta localizar a referência a biblioteca e descomentar a linha retirando o caractere #.

Listagem 1. ContasPagar e ContasReceber

```
<?php
//Contas Receber
class ContasReceber{
     private $id_contasreceber = null;
     private $documento_contasreceber = null;
    private $valor_contasreceber = null;
    private $cliente_contasreceber = null;
    private $status_contasreceber = null;
    private $vencimento_contasreceber = null;

     public function getCliente_contasreceber(){
       return $this->cliente_contasreceber;
     }
    public function getDocumento_contasreceber(){
        return $this->documento_contasreceber;
     }
     public function getId_contasreceber(){
        return $this->id_contasreceber;
     }
    public function getStatus_contasreceber(){
        return $this->status_contasreceber;
     }
    public function getValor_contasreceber({
        return $this->valor_contasreceber;
     }
     public function getVencimento_contasreceber(){
        return $this->vencimento_contasreceber;
     }
    public function setCliente_contasreceber($cliente_
contasreceber){
        $this->cliente_contasreceber = $cliente_contasreceber;
     }
    public function setDocumento_contasreceber($documento_
contasreceber){
        $this->documento_contasreceber = $documento_contasreceber;
     }
    public function setId_contasreceber($id_contasreceber){
        $this->id_contasreceber = $id_contasreceber;
     }
    public function setStatus_contasreceber($status_contasreceber)
{
        $this->status_contasreceber = $status_contasreceber;
    }
    public function setValor_contasreceber($valor_contasreceber){
        $this->valor_contasreceber = $valor_contasreceber;
     }
    public function setVencimento_contasreceber($vencimento_
contasreceber){
        $this->vencimento_contasreceber = $vencimento_
contasreceber;
    }
}
?>
<?php
//Contas Pagar
class ContasPagar{
     private $id_contaspagar = null;
     private $documento_contaspagar = null;
     private $valor_contaspagar = null;
     private $fornecedor_contaspagar = null;
     private $status_contaspagar = null;
     private $vencimento_contaspagar = null;
    public function getDocumento_contaspagar(){
        return $this->documento_contaspagar;
     }
    public function getFornecedor_contaspagar(){
        return $this->fornecedor_contaspagar;
     }
    public function getId_contaspagar(){
        return $this->id_contaspagar;
     }
    public function getStatus_contaspagar(){
        return $this->status_contaspagar;
     }
    public function getValor_contaspagar(){
        return $this->valor_contaspagar;
     }
     public function getVencimento_contaspagar(){
        return $this->vencimento_contaspagar;
     }
    public function setDocumento_contaspagar($documento_
contaspagar){
        $this->documento_contaspagar = $documento_contaspagar;
     }
     public function setFornecedor_contaspagar($fornecedor_
contaspagar){
        $this->fornecedor_contaspagar = $fornecedor_contaspagar;
     }
    public function setId_contaspagar($id_contaspagar){
        $this->id_contaspagar = $id_contaspagar;
     }
    public function setStatus_contaspagar($status_contaspagar){
        $this->status_contaspagar = $status_contaspagar;
     }
    public function setValor_contaspagar($valor_contaspagar){
        $this->valor_contaspagar = $valor_contaspagar;
    }
     public function setVencimento_contaspagar($vencimento_
contaspagar){
        $this->vencimento_contaspagar = $vencimento_contaspagar;
    }
}
?>
```

Na realidade apenas teremos uma linha de código nesse arquivo PHP. Como visto logo a seguir. Para que não sabe quando uma variável é atribuída no PHP é a torna-se diponível a todos os scripts que fazem chamada ao script onde essa variável foi atribuída. Ou seja, usaremos o método require() ou require_once() toda vez que precisarmos usar tal variável.

```
$con = new mysqli("localhost", "root", "", "financeiro");
```

Criando as classes DAO

Vamos criar nesse momento as classes DAO (Data Access Objects), mas antes disso devemos entender como que funcionam essas classes. Data Access Object é um modelo de persistência de dados muito utilizando por desenvolvedores que desejam criar seus sistemas baseados em conceitos de orietação a objetos, como o MVC (Model – View – Controller), por exemplo. Em MVC todas as funções para persistência de dados e qualquer outra integração com o banco de dados devem se feitas em classe do tipo DAO. O principal objetivo disso é separar regras de negócios de regras de acesso a dados.

Nota do DevMan

Model-view-controller (MVC) é um padrão de arquitetura de software. Com o aumento da complexidade das aplicações desenvolvidas torna-se fundamental a separação entre os dados (Model) e o layout (View). Desta forma, alterações feitas no layout não afectam a manipulação de dados, e estes poderão ser reorganizados sem alterar o layout.

O model-view-controller resolve este problema através da separação das tarefas de acesso aos dados e lógica de negócio, lógica de apresentação e de interação com o utilizador, introduzindo um componente entre os dois: o Controller. MVC é usado em padrões de projeto de software, mas MVC abrange mais da arquitetura de uma aplicação do que é típico para um padrão de projeto.

Crie então um sub-diretório denominado DAO no diretório principal de nossa aplicação. Logo após vamos criar as duas classes citadas no diretório que foi criado. A DAO será responsável por conter os métodos de INSERT, SELECT, DELETE E UPDATE, em outras palavras: os métodos CRUD (Create, Read, Update e Delete). Com isso não vamos nos preocupar com os códigos SQL na hora de realizar alguma ação no banco de dados.

Como tinha mencionado anteriormente tudo será responsabilidade da nossa DAO, já que ela é a responsável pelo acesso a base de dados. Nas nossas telas de inclusão, edição, e outros, simplesmente vamos chamar seus respectivos métodos. Vamos iniciar primeiramente criando a ContasPagarDAO.php com os seguintes métodos: InsertContasPagar, DeleteContasPagar, ShowContasPagar, UpdateContasPagar, ShowFornecedores.

Pelos nomes já possível entender o que cada método fará. Você deve te percebido que indiquei a criação também de método chamado ShowFornecedores. Ele nada mais fará do que buscar todos fornecedores, precisamos disso simplesmente para quando realizarmos um cadastro ou edição do registro. Ele seja exibido em uma SelectList como na **Listagem 2**, onde temos todo o código das classes ContasPagarDAO.php e ContasReceberDAO.php.

Vamos entender como funciona a classe ContasPagarDAO. Como ambas as classe são bem semelhantes, vamos entender apenas ContasPagarDAO. ContasRecebeDAO encontra-se disponível no site da DevMedia para que todos possam fazer os devidos testes. Vejamos então.

Logo de início declaramos a função InsertContasPagar() que recebe como parâmetro a conta a ser paga. Na primeira linha da função encontramos uma chamada a variável $con que contém a conexão com o banco de dados. Logo em seguida preparamos uma instrução SQL para fazer a inclusão no banco. Passamos para a instrução, todos os dados necessários para a execução do script. Na sequência usamos o método bind_param() para informar o tipo de cada parâmetro informado na linha anterior. O método bind_param() recebe como parâmetro o tipo de cada dado e seus parâmetros, respectivamente. Perceba que usamos uma espécie de constante: sdiss. Isso significa que os parâmetros serão String, Double, Integer, String e String. Isso poupa código de conversão. Se quiser entender melhor como funciona, basta acessar o link http://br2.php.net/manual/pt_BR/mysqli-stmt.bind-param.php.

Agora vem a parte interssante do artigo. Nós recebemos um objeto do tipo ContasPagar no parâmetro de nossa function(). Isso porque agora teremos que usá-lo em conjunto com os métodos Get das respectivas classes. Precisamos agora informar quais valores precisam ser gravados no banco, veja o trecho a seguir:

```
$P1 = $ContasPagar->getDocumento_contaspagar();
$P2 = $ContasPagar->getValor_contaspagar();
$P3 = $ContasPagar->getFornecedor_contaspagar();
$P4 = $ContasPagar->getStatus_contaspagar();
$P5 = $ContasPagar->getVencimento_contaspagar();
```

Perceba que usamos os métodos getDocumento_contaspagar(), getValor_contaspagar(), getFornecedor_contaspagar(), getStatus_contaspagar(), getVencimento_contaspagar() que já nos é familiar. Por fim executamos a inclusão chamando o método Execute().

Não fechamos por aqui, ainda precisamos qual o resultado disso tudo. Portanto, é importante sabermos o quantos registros foram afetados pela instrução executada. Fazemos isso com uma chamada ao método affected_rows(). Ele deve nos retornar uma valor superior a zero (0) indicando que forma executados registros com sucesso.

O que vemos nas linhas subsequentes é bastante semelhante ao que acabamos de entender, porém as instruções SQL usadas fazem parte do restante do método CRUD. A função ShowContasPagar(), por exemplo, também recebe como parâmetro um objeto do tipo $ContaPagar. Novamente fazemos uma referência a variável $con para ter acesso ao banco de dados. Nós fazemos uma consulta ao banco de dados no intuito de receber o ID do registro, por isso usamos o método getId_contaspagar(). Caso não retorne nulo, montamos nosso retorno.

O método ShowFornecedores() apenas executa um loop no banco de dados para resgatar todos os fornecedores disponíveis. Perceba a instrução SQL que fazemos: SELECT * FROM FORNECEDORES. Depois retormamos o resultado em uma variável do tipo SelectList. DeleteContasPagar() e UpdateContasPagar() seguem basicamente as mesmas regras das funções já explicadas.

Como podemos ver não há nenhum grande mistério nisso tudo. A classe ContasReceberDAO() tem exatamente a mesma estrutura e propósito, porém concentrando suas operações nas respectivas tabelas.

Desenvolvendo as telas de cadastro

Até aqui temos todas as classe necessárias para o bom funcionamento do nosso sistema. Agora é o momento de trabalharmos com elas, para isso simplesmente vamos criar dois diretórios que conterão as telas de cadastro e consulta. Desenvolveremos algumas telas para que tenhamos uma noção de como trabalhar orietado a objetos com o PHP.

Vamos lá então. Crie duas pastas chamadas contaspagar e contasreceber na pasta principal do nosso Web Site. Dentro delas vão conter os arquivos listar.php, apagar.php, atualizar.php e cadastrar.php, todos em ambos diretórios. Para entendermos mais fácil, vamos desmembrar as listagens e ver os detalhes de cada uma. Na **Listagem 3** temos o código da tela responsável por listar nossos registros do Contas a Pagar. Digite o código da listagem indicada e salve-a como lista.php.

Entre as **Linhas 2** e **4** iniciamos o nosso script declarando os arquivos Conexao.php, ContasPagar.php e ContasPagarDAO.php. Esses foram os arquivos de desenvolvemos até o momento. O primeiro deles contém a variável $con com a conexão do banco de dados salva. Os demais arquivos são nossas classes.

Até a **Linha 21** como podemo notar, temos apenas código HTML que fará a impressão de uma tabela na tela. Na **Linha 22** até **27** estamos começando a utilizar nossas classes. Declarmos e atribuimos ao mesmo tempo três variáveis. A primeira delas recebe uma instância da classe ContasPagar() e a segunda a classe ContasPagarDAO(). Por fim recebemos os registros da tabela na variável $query (**Linha 25**). Na linha seguinte iniciamos um laço while para imprimir em tela os dados resultantes da SELECT efetuada pela classe ShowContasPagar(). Repare nas linhas seguintes que estamos mesclando código HTML com código PHP. Isso é bastante comum. Na **Linha 29**, por exemplo abrimos uma tag

Listagem 2. Código da classe ContasPagarDAO

```
<?php
class ContasPagarDAO{
     public function InsertContasPagar(ContasPagar $ContasPagar){
     global $con;
     $SQL = $con->prepare("INSERT INTO contaspagar (documento_
      contaspagar,
        valor_contaspagar, fornecedor_contaspagar, status_
       contaspagar,
        vencimento_contaspagar) VALUES (?, ?, ?, ?, ?)") or die
         ($mysqli->error);
     $SQL->bind_param("sdiss", $P1, $P2, $P3, $P4, $P5);

     $P1 = $ContasPagar->getDocumento_contaspagar();
     $P2 = $ContasPagar->getValor_contaspagar();
     $P3 = $ContasPagar->getFornecedor_contaspagar();
     $P4 = $ContasPagar->getStatus_contaspagar();
     $P5 = $ContasPagar->getVencimento_contaspagar();

     $SQL->execute();

     if ($SQL->affected_rows > 0){
        return true;
     }

     public function ShowContasPagar(ContasPagar $ContasPagar){
        global $con;
        if ($ContasPagar->getId_contaspagar() == null){
           $SQL = $con->query("SELECT * FROM contaspagar");
                return $SQL;
        }else{
           $SQL = $con->query("SELECT * FROM contaspagar WHERE
            id_contaspagar =
              '".$ContasPagar->getId_contaspagar()."'");
           $rs = $SQL->fetch_array();
           $ContasPagar->setId_contaspagar($rs["id_contaspagar"]);
           $ContasPagar->setDocumento_contaspagar($rs["documento_
             contaspagar"]);
           $ContasPagar->setFornecedor_
             contaspagar($rs["fornecedor_contaspagar"]);
           $ContasPagar->setValor_contaspagar($rs["valor_
             contaspagar"]);
           $ContasPagar->setVencimento_contaspagar
             ($rs["vencimento_contaspagar"]);
           $ContasPagar->setStatus_contaspagar($rs["status_
             contaspagar"]);
        }
     }
     public function ShowFornecedores(ContasPagar $ContasPagar){
        global $con;

        $SQL = $con->query("SELECT * FROM fornecedores");
        if ($ContasPagar->getId_contaspagar() == null){
           while($registros = $SQL->fetch_array()){

              $selectList[] = "<option
                 value='".$registros["id_fornecedor"]."'>".]
                 $registros["nome_fornecedor"]."</option>";
           }
        }else{
           $id = $ContasPagar->getFornecedor_contaspagar();
           while($registros = $SQL->fetch_array()){
              ($registros["id_fornecedor"] == $id ? 'selected=""' : '')

               $selectList[] = "<option ".
               ($registros["id_fornecedor"] == $id ? 'selected=""' : '').
               value='".$registros["id_fornecedor"]."'>".
               $registros["nome_fornecedor"]."</option>";
           }
        }
        return $selectList;
     }
     public function DeleteContasPagar(ContasPagar $ContasPagar){
        global $con;
        $SQL = $con->prepare("DELETE FROM contaspagar WHERE id_
         contaspagar = ?");
        $SQL->bind_param("i", $P1);
        $P1 = $ContasPagar->getId_contaspagar();;
        $SQL->execute();

        if($SQL->affected_rows > 0)
           return true;
     }

     public function UpdateContasPagar(ContasPagar $ContasPagar){
          global $con;
          $SQL = $con->prepare("UPDATE contaspagar SET documento_
           contaspagar = ?,
             valor_contaspagar = ?, fornecedor_contaspagar = ?,
             status_contaspagar = ?,  vencimento_contaspagar = ?
             WHERE id_contaspagar = ?");
          $SQL->bind_param("sdissi", $P1, $P2, $P3, $P4, $P5, $P6);
          $P1 = $ContasPagar->getDocumento_contaspagar();
          $P2 = $ContasPagar->getValor_contaspagar();
          $P3 = $ContasPagar->getFornecedor_contaspagar();
          $P4 = $ContasPagar->getStatus_contaspagar();
          $P5 = $ContasPagar->getVencimento_contaspagar();
          $P6 = $ContasPagar->getId_contaspagar();
          $SQL->execute();
          if($SQL->affected_rows > 0)
             return true;
     }
}
?>
```

Listagem 3. Código do arquivo listar.php

```
01 <?php
02 include "../Conexao.php";
03 include "../Classes/ContasPagar.php";
04 include "../DAO/ContasPagarDAO.php";
05 ?>
06
07 <head>
08 <style type="text/css">
08 @import url(../padrao.css);
10 </style>
11 </head>
12
13 <h1>Lista de Contas a Pagar</h1>
14 <hr/>
15 <table class="ms-classic3-main" style="width: 77%">
16    <tr>
17      <td class="ms-classic3-tl" style="width:
          209px">Documento:</td>
18      <td class="ms-classic3-top" style="width: 165px">Valor:</td>
19      <td class="ms-classic3-top" style="width: 160px">Status</td>
20      <td class="ms-classic3-top" style="width: 66px">Editar</td>
21   </tr>
22    <?php
23       $CP = new ContasPagar();
24      $CPDAO = new ContasPagarDAO();
25      $query = $CPDAO->ShowContasPagar($CP);
26      while($reg = $query->fetch_array()){
27    ?>
28  <tr>
29      <td class="ms-classic3-left" style="width:209px">
30         <?=$reg["documento_contaspagar"];?></td>
31      <td class="ms-classic3-even" style="width: 165px">
32          <?=$reg["valor_contaspagar"];?></td>
33      <td class="ms-classic3-even" style="width: 160px">
34          <?=$reg["status_contaspagar"];?></td>
35      <td class="ms-classic3-even" style="width: 66px">
36        <center>
37            <a href="atualizar.php?ID=<?=$reg["id_contaspagar"];?>">
38              <img alt="" height="16" src="../imagens/editar.
                  png" width="16" class="style1"
39                border="0"></a>
40            <a href="apagar.php?ID=<?=$reg["id_contaspagar"];?>">
41              <img alt="" height="16" src="../imagens/apagar.
                  png" width="16" class="style1"
42                border="0"></a>
43         </center>
44      </td>
45   </tr>
46    <?php } ?>
47 </table>
```

<td>, célula, e na **Linha 30** imprimimos o valor que está contido no array $reg iniciado e alimentado pelo laço while. Na sequência imprimimos o restante dos dados usando o mesmo método: HTML + PHP até que os registros cheguem ao final. Repare agora na **Linha 46**. Ela contém apenas a chave de fechamento do laço while, isso é muito importante de ser observado.

Uma dica importante que passo a todos, é sempre fazer primeiro a parte de design na página, depois implementar em seu código as regras de negócios, ou seja, o código que vai buscar no banco dados aquilo que precisa ser impresso. Dessa forma fica mais fácil gerenciar e finalizar com qualidade o código de qualquer Web Site.

O próximo arquivo a vermos nessa etapa de desenvolvimento dos layouts, é o arquivo que terá a responsabilidade de excluir o registro do Contas a Pagar. O script precisa também adicionar os arquivos Conexao.php, ContasPagar.php e ContasPagarDAO.php no início do código para que tenhamos acesso as classes e conexões com o banco. O script certamente é um dos mais simples até aqui. Primeiro verificamos se estamos recebendo a QueryString ID. Para quem não sabe, QueryString são os parâmetros recebidos na url de do browser, por exemplo:

www.meusite.com.br/cadastro.php?ID=1203130

Perceba que após o nome do arquivo cadastro.php informamos um ponto de interrogação e o parâmetro ID. Se a url estive digitada até cadastro.php, significa que a função isset() nos retornaria nulo para o parâmetro ID. Voltando ao script, havendo o parâmetro criamos uma instância da classe ContasPagar(), enviamos o ID para ela e criamos uma instância de ContasPagarDAO(). Feito isso, chamamos o método DeleteContasPagar() passando como parâmetro o objeto que fará a referência para excluir o registro desejado (**Linha 9**). Por fim exibimos a mensagem de confirmação (**Linha 10**). Não esqueça de salvar o script como apagar.php no diretório correto (**Listagem 4**).

Listagem 4. Código do script apagar.php

```
01 <?php
02 include "../Conexao.php";
03 include "../Classes/ContasPagar.php";
04 include "../DAO/ContasPagarDAO.php";
05 if (isset($_GET['ID'])){
06      $CP = new ContasPagar();
07      $CP->setId_contaspagar($_GET["ID"]);
08      $CPDAO = new ContasPagarDAO();
09      if ($CPDAO->DeleteContasPagar($CP)){
10         echo "<script>alert('Conta a pagar, deletada com
            succeso!');</script>";
11      }
12 }
13 ?>
```

Estamos quase fechando nosso sistema. Veremos agora como funciona a atualização de registros do contas a pagar. Crie um novo arquivo chamado atualizar.php. Esse script é necessário para que possamos dar a oportunidade ao usuário de alterar um registro do contas a pagar.

Nota

Para poupar espaço na revista e proporcionar maior entendimento do arquivo e dos código, omitimos alguns trechos de código que podem ser obtidos diretamene no site da DevMedia. Os trechos foram substituidos por [...] indicando que nesse pedaço de script há mais código. Os códigos omitidos não invalidam o artigo, pois são trecho HTML que podem facilmente se incorporados a qualquer momento.

O código da **Listagem 5** é um pouco mais extenso, portanto omitimos alguns trechos menos importantes para facilitar mais o aprendizado. Vamos as partes mais importantes do script. Novamente invocamos os arquivos necessários para ter acesso as classes e conexão com o banco. Na sequência, instanciamos a classe ContasPagar() e invocamos o método setId_contaspagar() para receber o ID da conta que vamos alterar. Criamos uma instância de ContasPagarDAO() e verificamos se existe o parâmetro atualizar na QueryString. Existindo, gravamos os dados necessários na tabela usando os métodos Set da classe (**Linhas 11** a **15**). Por último chamamos o método UpdateContasPagar() passando o objeto $CP que na verdade é uma referência ao objeto ContasPagar recebido na **Linha 6**. No final exibimos uma mensagem informando que ação teve sucesso.

A primeira parte desse script é bem simples, mas não termina por ai. Vejamos o que mais estamos fazendo para atualizar esses dados. Na **Linha 23** chamamos novamente o método ShowContasPagar() para que possamos obter novamente os dados do BD. Faremos a impressão dessas informações na tela.

Mais uma vez estamos mesclando código HTML e PHP. Criamos um formulário na **Linha 27**. Repare que o action do formulário está sendo apontado parar o arquivo atualizar.php, ou seja, o arquivo de script que estamos desenvolvendo nesse momento. Estamos enviando como parêmtro o ID do Contas a Pagar, lembra da história do QueryString? Pois bem, é nesse momento que a url é alterada e a primeira parte do script é executada.

Da **Linha 27** até o final fiz questão de manter, pois é a parte principal desse script. Veja nas **Linhas 31**, **38**, **46** a **50**, **55** e **63** que temos código PHP. Basicamente o código vem dentro de tags <td>, células, ou seja estamos imprimindo em tela os dados do registro que selecionamos, nada mais. No final ainda temos a inclusão de um botão para confirmar a atualização dos dados.

Enfim, veremos a última parte de nossos scripts: o cadastro. O cadastro deve nos permitir incluir novos registros de Contas a Pagar. O início do script contém o mesmo que fizemos até agora, ou seja, inclusão do link com o arquivos de classes e conexão. Já deu para perceber que precisaremos fazer isso sempre certo? Se fizermos uma comparação com Delphi ou .net, digamos que seria o mesmo que fazer um uses (Delphi) ou declararmos o namespace (.net) que contém as funções que usaremos.

Nosso script é novamente divido em dois assim como no script anterior. A primeira parte consiste verificar se existe a QueryString cadastro, havendo, instanciamos a classe ContasPagar() (**Linha 7**), setamos o conteúdo de cada campo, instanciamos ContasPagarDAO() (**Linha 13**) e chamamos o métod InsertContasPagar(). Esse procedimento todo será chamado quando o usuário clicar no botão para confirmar a inclusão do registro, como já veremos.

Novametne omiti algumas parte do código para que possamos entender mais facilmente. Lembre-se de baixar o exemplo completo no site da edição.

Na **Linha 21** estamos novamente criado um formulário e passando como parâmetro para a QueryString a string cadastro. Ela é usada na primeira parte desse script como já mencionado.

Listagem 5. Código do script atualizar.php

```
01 <?php
02 [...]
03 include "../Conexao.php";
04 include "../Classes/ContasPagar.php";
05 include "../DAO/ContasPagarDAO.php";
06 $CP = new ContasPagar();
07 $CP->setId_contaspagar($_GET["ID"]);
08 $CPDAO = new ContasPagarDAO();
09 $CP->setFornecedor_contaspagar($CP->getFornecedor_contaspagar());
10 if (isset($_GET['atualizar'])){
11     $CP->setDocumento_contaspagar($_POST['txtDocumento']);
12     $CP->setValor_contaspagar($_POST['txtValor']);
13     $CP->setFornecedor_contaspagar($_POST['cbFornecedor']);
14     $CP->setVencimento_contaspagar($_POST['txtData']);
15     $CP->setStatus_contaspagar($_POST['txtStatus']);
16     if ($CPDAO->UpdateContasPagar($CP)){
17        echo "<script>alert('Conta a pagar, atualizada com
           succeso!');</script>";
18     }
19 }
20 ?>
21 [...]
22 <?php
23      $CPDAO->ShowContasPagar($CP);
24 ?>
25 [...]
26 <form action="atualizar.php?atualizar&ID=<?=$_GET["ID"];?>"
    method="post">
27    <table style="width: 100%" class="ms-classic3-main">
28       <tr>
29          <td style="width: 136px" class="ms-classic3-tl">
             Documento:</td>
30          <td class="ms-classic3-top"><input name="txtDocumento"
              value="
31             <?=$CP->getDocumento_contaspagar();?>"
32             style="width: 292px" type="text" />
33          </td>
34       </tr>
35       <tr>
36          <td style="width: 136px" class="ms-classic3-left">
              Valor:</td>
37          <td class="ms-classic3-even"><input name="txtValor" value="
38             <?=$CP->getValor_contaspagar();?>"
39             style="width: 292px" type="text" />
40          </td>
41       </tr>
42       <tr>
43          <td style="width: 136px" class="ms-classic3-left">
             Fornecedor:</td>
44          <td class="ms-classic3-even">
45             <select name="cbFornecedor">
46                <?php
47                   foreach($CPDAO->ShowFornecedores($CP) as $exibir){
48                      echo $exibir;
49                   }
50                ?>
51             </select>
52          </td>
53       </tr>
54       <tr>
55          <td style="width: 136px" class="ms-classic3-left">Data
              de Vencimento:</td>
54         <td class="ms-classic3-even"><input name="txtData" value="
55             <?=$CP->getVencimento_contaspagar();?>"
56             style="width: 127px" type="text" />
57          </td>
58       </tr>
59       <tr>
60          <td style="width: 136px" class="ms-classic3-left">
              Status:</td>
61          <td class="ms-classic3-even">
62             <input name="txtStatus" value="
63                <?=$CP->getStatus_contaspagar();?>"
64                style="width: 127px" type="text" /> (S =
                   Positivo(Pago), N =
65                   Negativo)
66          </td>
67       </tr>
68       <tr>
69          <td style="width: 136px" class="ms-classic3-left"> </td>
70          <td class="ms-classic3-even"><input name="btCadastrar"
              type="submit"
71             value="Atualizar" />
72          </td>
73       </tr>
74    </table>
75 </form>
76 </body>
77 </html>
```

Em seguida criamos uma tabela com a tag <table> onde será impresso o registro que queremos incluir no banco. Repare nas **Linhas 25 a 31** que estamos instanciando o ContasPagar() e ContasPagarDAO() para depois fazermos um laço foreach() em fornecedores e imprimir em tela. Por fim, completamos o nosso formulário com um botão para confirmar o cadastro. Tudo isso estamos fazendo na **Listagem 6** que deverá ser salva como cadastro.php.

Não estamos completamente prontos, já que é necessário criar as telas de manutenção do Contas a Receber. Como os código são muito semelhantes, não veremos aqui. O ideal é que baixe o código-fonte completo no site da DevMedia e teste-o, pois o objetivo desse artigo é mostrar os principais pontos para que seja possível entender e reproduzir os exemplo mostrado.

Você pode ver parte do sistema funcionando nas **Figuras 2 e 3**. Como mencionei no início do artigo, o tema POO e muito complexo e detalhado. O recomendável é que recorra a outros artigos e até a livros especializados em conceitos para aprofundar-se ao máximo no tema.

Listagem 6. Código do arquivo cadastro.php

```
01 [...]
02 <?php
03 inlude "../Conexao.php";
04 include "../Classes/ContasPagar.php";
05 include "../DAO/ContasPagarDAO.php";
06 if (isset($_GET['cadastro'])){
07      $CP = new ContasPagar();
08      $CP->setDocumento_contaspagar($_POST['txtDocumento']);
09      $CP->setValor_contaspagar($_POST['txtValor']);
10      $CP->setFornecedor_contaspagar($_POST['cbFornecedor']);
11      $CP->setVencimento_contaspagar($_POST['txtData']);
12      $CP->setStatus_contaspagar("N");
13      $CPDAO = new ContasPagarDAO();
14      if ($CPDAO->InsertContasPagar($CP)){
15         echo "<script>alert('Conta a pagar, cadastrada com
            succeso!');</script>";
16         echo "<script>window.location = 'listar.php';</script>";
17      }
18 }
19 ?>
20 [...]
21 <form action="?cadastro" method="post">
22    <table style="width: 100%" class="ms-classic3-main">
23 [...]
24             <select name="cbFornecedor">
25                <?php
26                      $CP = new ContasPagar();
27                      $CPDAO = new ContasPagarDAO();
28                      foreach($CPDAO->ShowFornecedores($CP) as
                          $exibir){
29                         echo $exibir;
30                      }
31                ?>
32             </select>
33          </td>
34       </tr>
35       <tr>
36          <td style="width: 136px" class="ms-classic3-left">Data
              de Vencimento:</td>
37          <td class="ms-classic3-even"><input name="txtData"
              style="width: 127px"
38             type="text" /></td>
39       </tr>
40       <tr>
41       <td style="width: 136px" class="ms-classic3-left"> </td>
42       <td class="ms-classic3-even"><input name="btCadastrar"
          type="submit"
43          value="Cadastrar" /></td>
44       </tr>
45    </table>
46 </form>
47 </body>
48 </html>
```

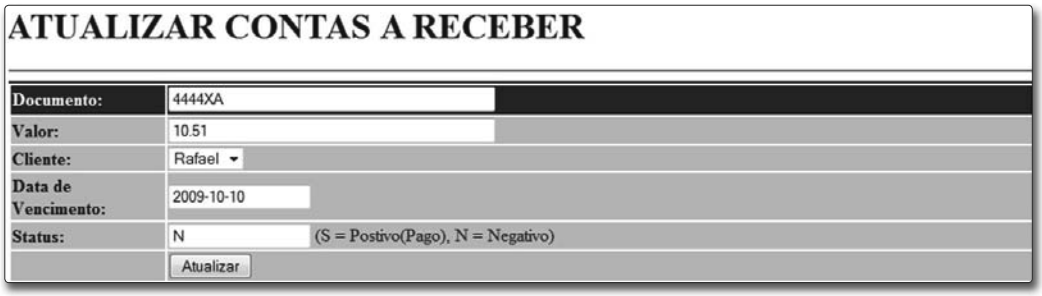


Figura 2. Atualização do Contas a Pagar

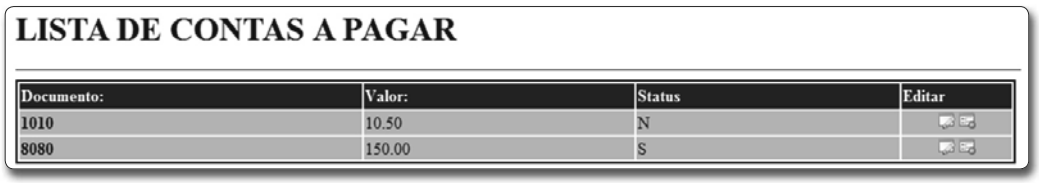


Figura 3. Lista de Contas a Pagar

Conclusão

Finalizamos nosso artigo com chave de ouro. Uma boa parte dos conceitos da POO puderam ser abordados ao longo de todo artigo. Pudemos observar de perto como criar uma classe DAO e usá-la em conjunto com MySQLI. Procurei ser o mais objetivo possível para que todo o conteúdo pudesse ser entendido. Não pudemos ver a segurança da aplicação, infelizmente. Isso vai ficar como lição de casa para os companheiros. Porém, quem tiver interesse em aprofundar-se ainda mais em POO ou mesmo em segurança de aplicações, há diversos artigos no próprio site da DevMedia.

Como dito diversas vezes, a POO é um tema bastante extenso e somento como muito estudo e treino é possível entendermos a fundo. Recomendo a leitura de outros artigos e livros especializados no tema.

De qualquer forma estou a disposição de todos para quaisquer esclarecimentos. Um grande abraço e até a próxima. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Existem momentos onde a estrutura faz toda a diferença.
Se o tempo é ruim, você encontra na Alog um porto seguro. Isto porque sua empresa reduz custos e melhora a governança em TI. Entre em contato e descubra porque a Alog é líder em hosting gerenciado.
REDUÇÃO DOS CUSTOS
AUMENTO DA GOVERNANÇA EM TI
guadalupe
0800 282 3330 | alog.com.br
ALOG
DATA CENTERS DO BRASIL
HOSTING GERENCIADO | COLOCATION | CONTINGÊNCIA | E-MAIL CORPORATIVO | CONECTIVIDADE | SERVIÇOS PROFISSIONAIS

VOCÊ levou anos para desenvolver, testar e colocar seu produto no mercado e **ELE** apenas alguns minutos para **PIRATEÁ-LO.**

SafeNet tem à sua disposição a proteção de software mais avançada e segura. As Chaves de Hardware Sentinel oferecem tecnologia que incorpora criptografia de chave pública, criptografia AES e autenticação interna para garantir o mais alto nível de proteção anti-pirataria.

Peça seu Kit de Desenvolvimento hoje mesmo.

Mais informações www.safenet-inc.com/antipirataria/03
e receba o Estudo de Caso.

SafeNet®
The Foundation of Information Security